



TESIS - KI142502

PEMBENTUKAN DATA UJI MENGGUNAKAN ALGORITMA OPTIMISASI KOLONI SEMUT DAN PENDEKATAN TEKNIK PENGUJIAN KOTAK ABU- ABU

SISKA ARIFIANI
5114201043

DOSEN PEMBIMBING
Dr. Ir. Siti Rochimah, MT.
NIP. 196810021994032001

PROGRAM MAGISTER
BIDANG KEAHLIAN REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

[Halaman ini sengaja dikosongkan]



THESIS - KI142502

GENERATING TEST DATA USING ANT COLONY OPTIMIZATION (ACO) ALGORITHM AND GRAY BOX TESTING APPROACH

SISKA ARIFIANI
5114201043

SUPERVISOR
Dr. Ir. Siti Rochimah, MT.
NIP. 196810021994032001

MASTER PROGRAM
DEPARTEMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

[Halaman ini sengaja dikosongkan]

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:

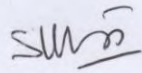
SISKA ARIFIANI
NRP. 5114201043

Dengan judul :
PEMBENTUKAN DATA UJI MENGGUNAKAN ALGORITMA OPTIMISASI KOLONI
SEMUT DAN PENDEKATAN TEKNIK PENGUJIAN KOTAK ABU-ABU

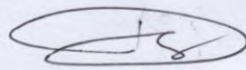
Tanggal Ujian : 24 Juni 2016
Periode Wisuda : 2015 Genap

Disetujui oleh:

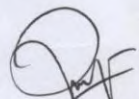
Dr. Ir. Siti Rochimah, M.T
NIP. 196810021994032001


(Pembimbing 1)

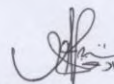
Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng.
NIP. 197411232006041001


(Penguji 1)

Rizky Januar Akbar, S.Kom, M.Eng
NIP. 198701032014041001

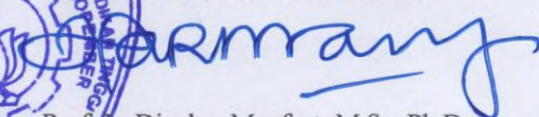

(Penguji 2)

Adhatus Sholichah, S.Kom, M.Sc
NIP. 198508262015042002


(Penguji 3)



Direktur Program Pasca Sarjana,


Prof. Ir. Djauhar Manfaat, M.Sc, Ph.D
NIP. 196012021987011001

[Halaman ini sengaja dikosongkan]

Pembentukan Data Uji Menggunakan Algoritma Optimisasi Koloni Semut dan Pendekatan Teknik Pengujian Kotak Abu-Abu

Nama Mahasiswa : Siska Arifiani
NRP : 5114 201 043
Pembimbing : Dr. Ir. Siti Rochimah, MT.

ABSTRAK

Pengujian perangkat lunak dapat menjadi cara yang efektif untuk memperbaiki kualitas serta ketahanan perangkat lunak. Secara umum pengujian perangkat lunak dapat dilakukan dengan teknik kotak putih (*white box*) dan kotak hitam (*black box*). Teknik pengujian kotak abu-abu (*gray box*) dikenal sebagai teknik pengujian yang menggunakan pendekatan teknik pengujian kotak putih dan kotak hitam. Teknik pengujian kotak abu-abu memperluas kriteria cakupan logika dari teknik pengujian kotak putih dan menemukan semua kemungkinannya dari model desain perangkat lunak sepertipada teknik pengujian kotak hitam. Teknik pengujian kotak abu-abu menggunakan data seperti: UML Diagram, Model Arsitektur perangkat lunak, ataupun Finite State Machine Diagram (State Model) untuk membentuk kasus pengujian.

Selain itu pengujian perangkat lunak dapat dilakukan dengan teknik Soft Computing dan Hard Computing. Teknik Hard Computing sulit diimplementasikan pada problematika yang ada saat ini. Sehingga teknik Soft Computing dapat menjadi alternatif yang digunakan dalam pengujian perangkat lunak. Teknik Soft Computing merupakan teknik yang lebih berfokus pada menginterpretasikan perilaku sistem dari pada hasil presisi. Teknik Soft Computing biasanya berdasarkan teknik logika fuzzy, jaringan saraf tiruan ataupun pengambilan keputusan berdasarkan nilai distribusi kemungkinan. Teknik pengujian perangkat lunak yang menggunakan teknik Soft Computing dengan metode pengambilan sampel berdasarkan nilai kemungkinan dikenal dengan teknik pengujian statistika. Berdasarkan penelitian terkait, algoritma Ant Colony Optimization (ACO) atau optimisasi koloni semut merupakan algoritma yang digunakan pada teknik pengujian statistika untuk membentuk data uji dengan kemampuan yang lebih baik dibandingkan algoritma lain seperti: Simulated Annealing (SA) serta algoritma genetika. Selain itu ACO juga memiliki hasil yang sebanding dengan algoritma Particle Swarm Optimization (PSO) atau optimisasi kawanan partikel. ACO diimplementasikan pada kode program perangkat lunak yang diuji untuk membentuk data uji berdasarkan nilai kemungkinan terbesar random data uji dari domain terpilih.

Pemilihan data uji merupakan faktor utama yang menentukan keberhasilan dari suatu pengujian perangkat lunak. Sehingga pemilihan teknik yang tepat dapat membantu menunjang keberhasilan dalam pengujian perangkat lunak. Pada penelitian ini, ACO diimplementasikan berdasarkan teknik pengujian kotak abu-abu menggunakan diagram UML State Machine. Pembentukan data uji yang berkualitas adalah berdasarkan kecukupan kriteria percabangan yang dapat ditelusuri. Tujuan dari penelitian ini adalah untuk mendapatkan hasil perbandingan pembentukan data uji dengan teknik pengujian kotak abu-abu menggunakan diagram UML State Machine dan teknik pengujian struktur kotak putih menggunakan kode program. Hasil penelitian ini diharapkan mampu memberikan gambaran kualitas data uji yang dibentuk dari masing-masing teknik.

Kata Kunci: Pengujian perangkat lunak, Teknik pengujian statistika, Optimisasi Koloni Semut.

[Halaman ini sengaja dikosongkan]

Generating Test Data using Ant Colony Optimization (ACO) Algorithm and Gray Box Testing Approach

Student Name : Siska Arifiani
NRP : 5114 201 043
Supervisor : Dr. Ir. Siti Rochimah, MT.

ABSTRACT

Software testing can be an effective way to improve software quality and reliability. In general, software testing techniques can be classified into White Box and Black Box. Gray Box testing technique is known as testing technique which is used both White Box and Black Box techniques. It extends the logical coverage criteria of white box method and finds all the possibility from the design model which is like black box method. Gray Box technique uses data such as: UML Diagram, Software Architecture Model, or Finite State Machine Diagram (State Model) to generate test cases.

The other classification of software testing technique is Hard Computing technique and Soft Computing technique. Hard Computing technique is difficult to be implemented in today problems. And Soft Computing technique can be an alternative way which can be used in software testing. Soft Computing technique focuses on system behavior interpretation than precision result. Soft Computing technique is based on fuzzy logic, neural network, or decision making by probability distribution. Software testing which is used Soft Computing technique and its sampling method based on probability distribution, is known as statistical testing. Based on research, Ant Colony Optimization (ACO) Algorithm is algorithm which is used in statistical testing to generate test data, and its result better than another algorithm such as: Simulated Annealing (SA), and Genetic algorithm. Besides, its result is comparable with Paricle Swarm Optimization (PSO) algorithm. ACO was implemented in program code of software under test to generate test data based on the highest probability value of random test data from domain chosen.

Test data selection is main factor which determines the software testing success. In this research, ACO was implemented based on Gray Box testing using UML State Machine Diagram. The quality of test data generated is based on branch coverage criteria. This research aims to get comparison result between Gray Box testing using UML State Machine Diagram and structural White Box Testing using program code in generating test data. The result of this research is expected to give description about the quality of test data generated from each technique.

Keywords: Software Testing, Gray Box Testing, Software Statistical Testing, Ant Colony Optimization, UML State Machine Diagram.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR



Segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis bisa menyelesaikan Tesis yang berjudul “Pembentukan Data Uji Menggunakan Algoritma Optimisasi Koloni Semut Dan Pendekatan Teknik Pengujian Kotak Abu-Abu” sesuai dengan target waktu yang diharapkan.

Pengerjaan Tesis ini merupakan suatu kesempatan yang sangat berharga bagi penulis untuk belajar memperdalam ilmu pengetahuan. terselesaikannya buku Tesis ini, tidak terlepas dari bantuan dan dukungan semua pihak. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. Allah SWT atas limpahan rahmat-Nya sehingga penulis dapat menyelesaikan Tesis ini dengan baik.
2. Ibu Nuraini dan Almarhum Bapak Moh. Arif, selaku orang tua penulis yang selalu mendoakan agar selalu diberikan kelancaran dan kemudahan dalam menyelesaikan Tesis ini. Serta menjadi motivasi terbesar untuk mendapatkan hasil yang terbaik.
3. Ibu Dr. Ir. Siti Rochimah, M.T. selaku dosen pembimbing yang telah memberikan kepercayaan, motivasi, bimbingan, nasehat, perhatian serta semua bantuan yang telah diberikan kepada penulis dalam menyelesaikan Tesis ini.
4. Bapak Daniel Oranova Siahaan, S.Kom, M.Sc, PD.Eng, Bapak Rizky Januar Akbar, S.Kom, M.Eng, Ibu Adhatus Sholichah, S.Kom, M.Sc dan Ibu Nurul Fajrin Ariyani, S.Kom, M.Sc. selaku dosen penguji yang telah memberikan bimbingan, saran, arahan, dan koreksi dalam pengerjaan Tesis ini.
5. Bapak Waskitho Wibisono, S.Kom., M.Eng., PhD selaku ketua program pascasarjana Teknik Informatika ITS, Bapak Dr. Ir. R.V. Hari Ginardi, M.Kom., selaku dosen wali penulis dan segenap dosen Teknik Informatika yang telah memberikan ilmunya.
6. Mbak Lina, Mas Kunto dan segenap staf Tata Usaha yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
7. Adik penulis, Septian Nur Ariyanto, serta seluruh keluarga besar yang selalu memberi semangat, do'a, dukungan dan hiburan kepada penulis.

8. Rekan seperjuangan Vika Fitratananny Insanittaqwa dan Felix Handani yang selalu menjadi teman diskusi yang baik, serta selalu mendo'akan dan mendukung penulis. Insya Allah rekan-rekan semua diberikan kemudahan dalam penyelesaian tesis.
9. Rekan-rekan angkatan 2014 Pasca Sarjana Teknik Informatika ITS yang telah menemani dan memberikan bantuan serta motivasi untuk segera menyelesaikan Tesis ini.
10. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu disini yang telah membantu terselesaikannya Tesis ini.

Sebagai manusia biasa, penulis menyadari bahwa Tesis ini masih jauh dari kesempurnaan dan memiliki banyak kekurangan. Sehingga dengan segala kerendahan hati, penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Surabaya, Juli 2016

DAFTAR ISI

ABSTRAK.....	vii
ABSTRACT.....	ix
KATA PENGANTAR	xi
DAFTAR ISI.....	xiii
DAFTAR GAMBAR	xvii
DAFTAR TABEL.....	xix
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Perumusan Masalah	3
1.3. Batasan Masalah.....	3
1.4. Tujuan	4
1.5. Manfaat Penelitian	4
1.6. Kontribusi Penelitian.....	4
BAB 2 DASAR TEORI DAN KAJIAN PUSTAKA	5
2.1. Verifikasi dan Validasi Perangkat Lunak	5
2.2. Teknik Pengujian Gray Box.....	6
2.3. UML State Machine Diagram.....	7
2.3.1. Pembentukan Kasus Uji Melalui UML State Machine Diagram dengan Pendekatan Prioritas	8
2.3.2. Pembentukan Kasus Uji Melalui UML State Machine Diagram dengan Kriteria Kecukupan Tertentu.....	9
2.3.3. Teknik Lain pada Pembentukan Kasus Pengujian dengan UML State Machine Diagram.....	10
2.4. Teknik Pengujian Statistika.....	13
2.4.1. Teknik Pembentukan Kasus Pengujian.....	15
2.4.1.1. Clustering	15
2.4.1.2. Markov Model.....	16
2.4.1.3. Teknik Pembentukan Kasus Pengujian Lainnya.....	17
2.4.2. Teknik Pembentukan Data Uji	17
2.4.2.1. Search Based	17
2.4.2.2. Huffman Code.....	19
2.4.2.3. Particle Swarm Optimization (PSO).....	19

2.4.2.4. Ant Colony Optimization (ACO).....	19
2.4.3. Teknik Pembentukan Data Uji Lainnya	20
2.5. Algoritma Ant Colony Optimization	20
BAB 3 METODOLOGI PENELITIAN	23
3.1. Tahapan Penelitian	23
3.2. Studi Literatur	23
3.3. Perancangan dan Implementasi.....	24
3.3.1 Perancangan	24
3.3.2 Implementasi	25
3.3.2.1 XMI <i>Creation</i>	28
3.3.2.2 XMI <i>Parsing</i>	29
3.3.2.3 Graph <i>Creation</i>	29
3.3.2.4 <i>Test PathCreation</i>	30
3.3.2.5 <i>Test CaseGenerator</i>	30
3.4. Analisis Pengujian.....	33
BAB 4 UJI COBA DAN EVALUASI.....	37
4.1. Implementasi Penelitian.....	37
4.2. Perancangan Uji Coba.....	37
4.2.1. Pembagian Data Set	37
4.2.2. Skenario Uji Coba.....	38
4.2.3. Implementasi Teknik Pengujian Kotak Abu-Abu.....	38
4.2.3.1. Proses UML Creation.....	39
4.2.3.2. Proses Diagram Validation	40
4.2.3.3. Proses XMI Creation.....	42
4.2.3.4. Proses XMI Parsing	42
4.2.3.5. Proses Graph Creation	42
4.2.3.6. Test Path Creator.....	43
4.2.3.7. Test Case Generator	44
4.3. Analisis Hasil	48
BAB 5PENUTUP	55
5.1. Kesimpulan	55
5.2. Saran.....	55
DAFTAR PUSTAKA	57

LAMPIRAN.....	63
BIODATA PENULIS	95

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

Gambar 2. 1. Teknik Pengujian Perangkat Lunak	6
Gambar 3. 1. Tahap Metodologi Penelitian	23
Gambar 3. 2. Tahapan Proses Implementasi.....	26
Gambar 3. 3. Form Survey untuk Validasi Diagram Manual	27
Gambar 3. 4. Tahapan Menu untuk Mengekspor File XMI	28
Gambar 3. 5. Dialog untuk Mengekspor File XMI.....	28
Gambar 3. 6. Hasil Ekspor File XMI.....	29
Gambar 3. 7. <i>Graph</i> yang Terbentuk	29
Gambar 3. 8. Kemungkinan Jalur Kasus Pengujian	30
Gambar 4. 1. Skenario Uji Coba.....	38
Gambar 4. 2. Kode Sumber Data Set Program GCD.....	39
Gambar 4. 3. Diagram UML State Machine dari Fungsi GCD	40
Gambar 4. 4. <i>Graph</i> yang Terbentuk	43
Gambar 4. 5. Alur yang Mungkin Berdasarkan Kecukupan Kriteria	43
Gambar 4. 6. Grafik Perbandingan Matriks AC pada Masing-Masing Teknik	50
Gambar 4. 7. Grafik Perbandingan Matriks SR pada Masing-Masing Teknik.....	51
Gambar 4. 8. Grafik Perbandingan Matriks AG pada Masing-Masing Teknik.....	52
Gambar 4. 9. Grafik Perbandingan Matriks AT pada Masing-Masing Teknik	53

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

Tabel 2. 1. Pembagian Teknik Pengujian Perangkat Lunak	14
Tabel 3. 1. Data Set yang Digunakan	24
Tabel 3. 2. Daftar Kolom pada Tabel Atribut.....	29
Tabel 3. 3. Kategori Kondisi untuk Menghitung Jarak Percabangan	32
Tabel 4. 1. Data Set Fungsi yang Digunakan.....	37
Tabel 4. 2. Hasil Survey Validasi Diagram GCD	41
Tabel 4. 3. Atribut pada Diagram UML State Machine	42
Tabel 4. 4. Kombinasi yang Terbentuk dari Domain Terdefinisi	45
Tabel 4. 5. 50 Semut yang Terpilih Secara Random	45
Tabel 4. 6. Jarak Percabangan untuk Masing-Masing Semut Random Terpilih.	46
Tabel 4. 7. Hasil Perhitungan Nilai Berat.....	47
Tabel 4. 8. Nilai Fitness untuk Random Ant Terpilih.....	47
Tabel 4. 9. Hasil Uji Coba Fungsi GCD	48
Tabel 4. 10. Hasil Matrik Pengukuran pada Teknik Pengujian Kotak Abu-Abu Menggunakan Diagram UML State Machine.....	49
Tabel 4. 11. Hasil Matriks Pengukuran pada Teknik Pengujian Kotak Putih Menggunakan Struktur Kode Program	49
Tabel 4. 12. Nilai <i>pvalue</i> untuk Matriks AC.....	51
Tabel 4. 13. Nilai <i>pvalue</i> untuk Matriks SR	52

[Halaman ini sengaja dikosongkan]

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Pengujian perangkat lunak adalah sebuah proses dimana sebuah perangkat lunak dianalisa kemampuannya berdasarkan kasus dan data uji tertentu. Perangkat lunak yang diuji dipelajari dan dibuat se-sistematik mungkin untuk dapat mengidentifikasi data uji, skenario pengujian, alur data, serta keadaan tertentu yang akan digunakan, agar nantinya pengujian yang dilakukan dapat dengan tepat mendeteksi kesalahan yang ada pada perangkat lunak tersebut.

Pengujian perangkat lunak dapat dikatakan sebagai fase yang paling kritis dan mahal dalam pengembangan perangkat lunak (Patwa & Malviya, 2010). The National Institute of Standart and Technology (NIST) melaporkan bahwa biaya yang dikeluarkan akibat kerusakan atau kecacatan sebuah perangkat lunak adalah \$59.9 milyar (dolar US) per tahunnya. Laporan tersebut mengestimasi bahwa 22 milyar dolar seharusnya tidak perlu dikeluarkan dengan menggunakan metode pendeteksian dini dari kerusakan sebuah perangkat lunak. Berdasarkan laporan ini pula, biaya yang dikeluarkan untuk proses identifikasi dan koreksi dari perangkat lunak yang rusak diperkirakan mencapai 80% dari total biaya pengembangan perangkat lunak seluruhnya (Khan & Kausar, 2013).

Secara umum, teknik pengujian dapat dilakukan secara kotak putih (white box) dan kotak hitam (black box). Kotak putih merupakan sebuah teknik pengujian yang menggunakan Software Under Test (SUT) atau perangkat lunak yang diuji sebagai petunjuk pengujian yang akan dilakukan. Teknik pengujian kotak putih bergantung pada analisa dampak perubahan dari kode program. Sehingga teknik pengujian kotak putih dapat dikatakan sebagai teknik untuk memvalidasi perubahan kode yang terjadi, tidak untuk memvalidasi spesifikasi kebutuhan dari suatu perangkat lunak. Sedangkan teknik pengujian kotak hitam merupakan teknik pengujian yang menguji level yang lebih tinggi dari suatu perangkat lunak seperti desain tampilan, ataupun spesifikasi kebutuhan. Teknik pengujian kotak hitam yang menggunakan spesifikasi kebutuhan untuk

membentuk sebuah kasus uji pengujian dikenal dengan Requirement Based Testing (RBT) atau pengujian berdasarkan kebutuhan. Dimana dalam RBT biasanya menggunakan diagram UML sebagai petunjuk untuk menentukan kasus uji.

Sebuah pendekatan yang dilakukan oleh Suppandee Sandhu dan Amardeep Singh (2011) pada penelitiannya adalah dengan menggunakan teknik pengujian kotak abu-abu. Yaitu sebuah teknik yang menggunakan baik teknik pengujian kotak putih dan kotak hitam. Dalam penelitian tersebut, kasus uji ditentukan dari permodelan diagram UML Activity untuk kemudian diujicobakan pada perangkat lunak. Teknik pengujian kotak abu-abu memperluas kriteria cakupan logika dari teknik pengujian kotak putih dan menemukan semua kemungkinannya dari model desain perangkat lunak seperti pada teknik pengujian kotak hitam. Teknik pengujian kotak abu-abu memiliki kelebihan dari teknik pengujian kotak hitam dan mengkombinasikannya dengan kode yang dimaksud dalam sistem seperti pada teknik pengujian kotak putih. Pada teknik pengujian ini, bahasa spesifikasi kebutuhan digunakan untuk lebih mudah mengerti kebutuhan dan memverifikasi kebenarannya melalui logika program (Priya & Sheba, 2013).

Pada penelitian ini, pendekatan teknik pengujian kotak abu-abu digunakan untuk membentuk kasus pengujian pada diagram UML State Machine. Berdasarkan penelitian yang dilakukan Michael Felderer dan Andrea Herrmann (2015) disimpulkan bahwa diagram UML State Machine dapat mendeteksi eror lebih banyak jika dibandingkan dengan diagram UML Activity. Beberapa literatur menyebutkan bahwa diagram UML State Machine merupakan contoh lain dari Finite Automata, Moore Machine, Finite State Machine, ataupun Petri Nets. Diagram UML State Machine disebut juga sebagai *directed graph*. Sehingga diagram UML State Machine dapat menggambarkan dengan baik struktur program dari perangkat lunak.

Untuk membentuk data uji yang optimal, penelitian ini menggunakan pendekatan statistika berupa algoritma Ant Colony Optimization (ACO) atau optimisasi koloni semut. Teknik pengujian pendekatan statistika adalah teknik pengujian yang menyusun data uji berdasarkan hasil pengambilan sampel dari distribusi probabilitas yang didefinisikan berdasarkan domain input perangkat

lunak. Teknik pengujian statistika ini diimplementasikan pada diagram UML State Machine, dan diverifikasi pada kode sumber SUT.

Penelitian Chengying Mao (2015) membandingkan algoritma ACO dengan beberapa algoritma seperti: algoritma pencarian (Simulated Annealing), algoritma Genetika, serta algoritma Particle Swarm Optimization (PSO) atau optimisasi kawanan partikel. Hasil penelitian tersebut menyebutkan bahwa algoritma ACO lebih baik jika dibandingkan algoritma lainnya. Dan algoritma ACO adalah sebanding dengan algoritma PSO.

Penelitian ini membandingkan teknik pengujian kotak putih pada struktur kode program dengan teknik kotak abu-abu menggunakan diagram UML State Machine. Teknik pengujian kotak abu-abu menggunakan Reverse Engineering atau rekayasa balik dari data set pada penelitian Chengying Mao (2015). Penelitian ini diharapkan dapat memperoleh hasil perbandingan antara teknik pengujian kotak putih pada struktur program dan teknik pengujian kotak abu-abu pada diagram UML State Machine.

1.2. Perumusan Masalah

Rumusan masalah pada penelitian ini adalah sebagai berikut:

1. Bagaimana membentuk data uji menggunakan diagram UML State Machine.
2. Bagaimana membentuk data uji dengan menggunakan algoritma ACO.
3. Bagaimana perbandingan hasil data uji yang terbentuk dari diagram UML State Machine dan struktur kode dengan menggunakan algoritma ACO.

1.3. Batasan Masalah

Permasalahan yang dibahas pada penelitian ini memiliki beberapa batasan sebagai berikut:

1. Data uji yang digunakan dalam penelitian ini berupa program dengan paradigma pemrograman prosedural yang terdiri dari minimal 10 baris hingga 300 baris.
2. Kriteria kecukupan percabangan dalam pemilihan data uji pada penelitian ini diukur dengan menggunakan matriks AC (Average Coverage), SR (Success Rate), AG (Average Generation), dan AT (Average Time) dari masing-masing metode yang digunakan.

3. Pada penelitian ini, uji coba masing-masing data uji pada program dibatasi hingga 5 kali uji coba.
4. Penetapan parameter-parameter inialisasi awal adalah berdasarkan pada penelitian sebelumnya (Mao, Xiao, Yu, & Chen, 2015).

1.4. Tujuan

Berdasarkan perumusan masalah yang dijelaskan pada subbab 1.2, penelitian ini bertujuan untuk memperoleh gambaran dari hasil perbandingan teknik pengujian kotak putih menggunakan struktur kode program dan teknik pengujian kotak abu-abu menggunakan diagram UML State Machine. Kedua teknik pengujian ini mengimplementasikan algoritma ACO untuk membentuk kasus pengujian berdasarkan pendekatan statistika.

1.5. Manfaat Penelitian

Adapun manfaat dari penelitian ini adalah :

1. Membantu memperoleh gambaran tahapan dalam memodelkan diagram UML State Machine menjadi rancangan pengujian untuk membentuk kasus uji pada pengujian perangkat lunak menggunakan teknik Soft Computing.
2. Dapat memperoleh gambaran hasil perbandingan dari pengujian pembentukan kasus uji menggunakan diagram UML State Machine dan struktur kode program dengan menggunakan Teknik Soft Computing.
3. Dapat membangun sebuah kakas bantu yang semi otomatis dalam membentuk kasus uji dan data uji pada suatu pengujian perangkat lunak.

1.6. Kontribusi Penelitian

Kontribusi yang diharapkan dari penelitian ini adalah dapat membuktikan bahwa pembentukan kasus uji dengan menggunakan diagram UML State Machine (teknik pengujian kotak abu-abu) dapat memperoleh hasil pengujian perangkat lunak yang tidak jauh berbeda atau sama jika dibandingkan dengan pembentukan kasus uji dengan menggunakan struktur kode SUT (teknik pengujian kotak putih). Selain itu, penelitian ini juga diharapkan mampu mengembangkan sebuah kakas bantu untuk membentuk kasus uji dan data uji pada suatu pengujian perangkat lunak secara semi otomatis dengan mengimplementasikan algoritma ACO.

BAB 2

DASAR TEORI DAN KAJIAN PUSTAKA

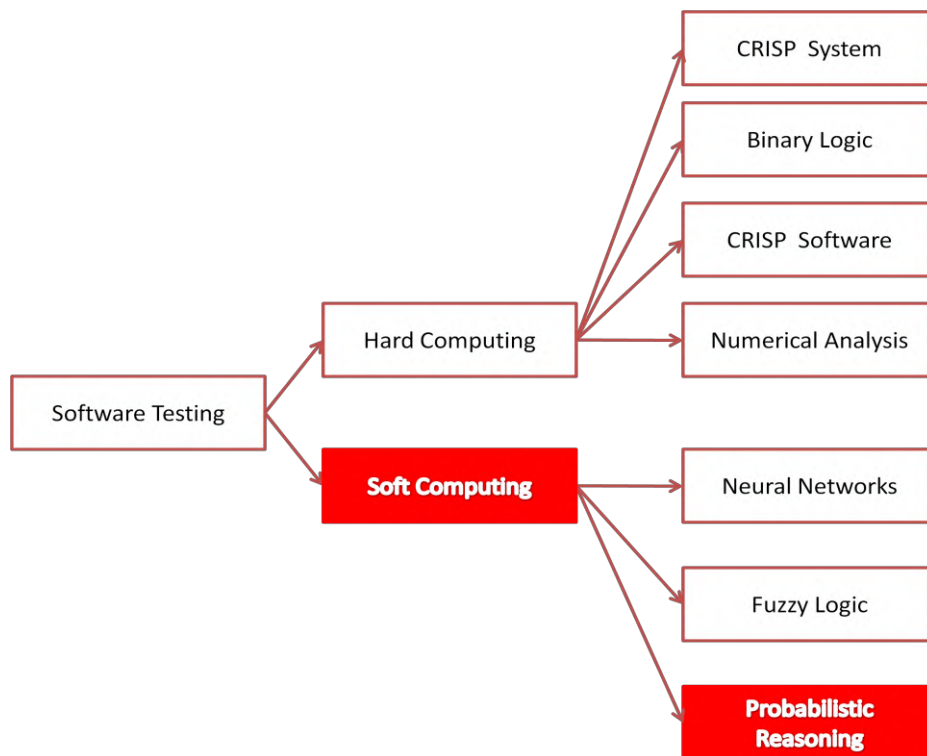
Pada bab ini dijelaskan dasar teori dan kajian pustaka yang terkait dengan penelitian.

2.1. Verifikasi dan Validasi Perangkat Lunak

Perangkat lunak dapat diukur ketahanannya melalui pengujian perangkat lunak, deteksi eror, bug dan kesalahan pada perangkat lunak ataupun metode lain yang dapat digunakan untuk memverifikasi dan validasi perangkat lunak. Penelitian ini adalah tentang pengujian perangkat lunak. Pengujian perangkat lunak dapat dilakukan dengan teknik Hard Computing dan Soft Computing. Teknik Hard Computing merupakan teknik komputasi konvensional yang membutuhkan model analisa yang presisi dan membutuhkan waktu komputasi yang lebih banyak. Sedangkan Soft Computing merupakan teknik yang lebih berfokus pada menginterpretasikan perilaku system dari pada hasil presisi. Teknik Hard Computing berdasarkan pada logika biner, analisa numerik, dan sistem CRISP (Computer Retrieval of Information on Scientific Projects). Sedangkan teknik Soft Computing berdasarkan pada logika fuzzy, jaringan saraf buatan, dan nilai kemungkinan. Gambar 2.1 menunjukkan pemetaan teknik-teknik tersebut.

Pengujian perangkat lunak itu sendiri secara umum terbagi menjadi teknik pengujian kotak putih dan kotak abu-abu. Penelitian ini menggunakan pendekatan teknik pengujian kotak abu-abu, yaitu teknik pengujian yang menggabungkan teknik pengujian kotak putih dan kotak hitam.

Penelitian ini membandingkan teknik pengujian perangkat lunak kotak putih menggunakan struktur kode program dengan teknik pengujian kotak abu-abu menggunakan diagram UML State Machine. Selain itu, penelitian ini menggunakan pendekatan teknik Soft Computing untuk membentuk data uji yang optimal kualitasnya. Pada Gambar 2.1 digambarkan bahwa teknik *probabilistic reasoning* atau pengambilan keputusan berdasarkan nilai kemungkinan digunakan sebagai teknik Soft Computing pada penelitian ini.



Gambar 2. 1. Teknik Pengujian Perangkat Lunak

2.2. Teknik Pengujian Gray Box

Teknik pengujian kotak abu-abu merupakan teknik pengujian yang mengkombinasikan teknik pengujian kotak hitam dan teknik pengujian kotak putih. Teknik pengujian kotak abu-abu menggunakan bahasa spesifikasi kebutuhan seperti pada pengujian kotak hitam dan memverifikasi kebenarannya berdasarkan struktur kode seperti pada pengujian kotak putih.

Kasus pengujian dengan menggunakan teknik pengujian kotak hitam dapat dibentuk dari awal spesifikasi kebutuhan perangkat lunak dilakukan. Atau dengan kata lain, dengan teknik pengujian ini, kasus uji dapat dibentuk sebelum seluruh kode program dari perangkat lunak tersebut selesai.

Pengujian kotak abu-abu biasa digunakan pada perangkat lunak berorientasi objek, dimana objek-objek tersebut adalah unit terpisah yang memiliki kode eksekusi atau data. Aplikasi yang menggunakan kotak abu-abu antara lain: Architectural Model atau model arsitektural, Unified Modeling Language (UML) Model, dan Finite State Machine (State Model). Teknik-teknik yang digunakan pada pengujian kotak abu-abu antara lain: *matrix testing* atau pengujian matriks untuk menyatakan laporan atau status proyek, *regression testing* atau pengujian

regresi untuk menyatakan apakah terjadi perubahan pada kasus uji yang baru dibuat, pattern testing atau pengujian pola untuk memverifikasi aplikasi yang baik untuk desain atau arsitektur dan pola, serta Orthogonal Array Testing yang digunakan sebagai bagian dari semua kemungkinan kombinasi.

Teknik kotak abu-abu dikembangkan lebih lanjut untuk memungkinkan pengujian untuk mendeteksi cacat dalam Service-Oriented Architecture (SOA). Teknik pengujian kotak putih tidak cocok jika digunakan pada sistem berbasis situs karena berhubungan langsung dengan internal kode program dari system yang diuji. Teknik pengujian kotak putih dapat digunakan pada mutasi pesan yang menghasilkan tes otomatis untuk array yang menangani pengecualian. Strategi ini berguna untuk mendorong teknik pengujian kotak abu-abu lebih memiliki hasil yang mendekati hasil teknik pengujian kotak putih.

2.3. UML State Machine Diagram

Diagram UML State Machine adalah sebuah diagram yang terdiri dari *state*, transisi (alur antar *state*), kejadian (*event*, yaitu sesuatu yang menyebabkan adanya transisi antar *state*) dan aktifitas (*activity*, yang menggambarkan reaksi dari adanya transisi). Diagram State Machine digunakan untuk menggambarkan *state* dari sebuah objek pada suatu sistem dan sebuah kejadian yang menyebabkan sebuah *state* berubah pada setiap respon yang terjadi. Pada diagram UML State Machine, sebuah node merepresentasikan *state* dan garis merepresentasikan transisi yang terjadi antar *state*. Pada diagram UML State Machine terdapat empat macam pemicu (*trigger*) terjadinya sebuah transisi antar *state*, yaitu: sinyal (*signal*), panggilan (*calls*), perubahan waktu, dan perubahan yang terjadi di dalam *state* itu sendiri. Sebuah diagram UML State Machine yang efisien, fleksibel, dan simple dapat merepresentasikan algoritma pada sistem dengan baik.

Manuj Anggarwal dan Sangeeta Sabharwal (2012) melakukan survey pada pembentukan kasus uji dari diagram UML State Machine. Pada literatur tersebut, pembentukan kasus uji dari diagram UML State Machine dibagi menjadi tiga kategori antara lain: pembentukan kasus uji dengan pendekatan prioritas, pembentukan kasus uji dengan kriteria kecukupan tertentu, dan teknik-teknik lain dalam pembentukan kasus uji dari diagram UML State Machine seperti: teknik

menggunakan alur kontrol dan data, teknik dalam mengatasi perubahan dalam suatu *state*, dan teknik yang termasuk di dalamnya teknik-teknik lain yang mungkin dalam pembentukan data uji.

2.3.1. Pembentukan Kasus Uji Melalui UML State Machine Diagram dengan Pendekatan Prioritas

Sabharwal dkk, (2011) telah mengusulkan sebuah pendekatan untuk membentuk prioritas dari sebuah skenario kasus uji yang terbentuk dengan mengidentifikasi pengelompokan dari garis (*path*) yang kritis menggunakan konsep dari alur informasi matriks (information flow/IF). Pada penelitian ini, struktur data yang digunakan berupa tumpukan (*stack*) dengan implementasi menggunakan algoritma Genetika (GA). Diagram State Machine dimodelkan kedalam bentuk *graph* yang disebut dengan State Dependency Graph (SDG).

Mohanty dkk, (2011) telah mengusulkan sebuah teknik untuk memprioritaskan kasus uji yang terbentuk dengan mengimplementasikan teknik pengujian regresi pada Component Based Software System (CBSS). Untuk merepresentasikan skenario interaksi antar komponen, diagram UML State Machine dibentuk menjadi Component Interaction Graph (CIG).

Weibleder (2011), menginvestigasi dampak dari tujuan prioritasasi pengujian jika dilakukan teknik prioritas pada: elemen yang paling jauh (Far Elements First/ FEF atau Far Elements Last/FEL), faktor percabangan yang banyak (High Branching Factor First/HBFF atau High Branching Factor Last HBFL), kondisi atomic yang banyak (Many Atomic Conditions First/MACF atau Many Atomic Conditions Last/MACL), rasio penugasan positif yang tinggi (High Positive Assignment Ratio First/HPARF atau High Positive Assignment Ratio Last/HPARL). Pada penelitian ini, penulis mendefinisikan tujuan prioritas pengujian yang berlawanan untuk masing-masing aspek tersebut dan membandiingkannya dengan teknik prioritas random (Random Prioritization/RP). Penulis menyimpulkan bahwa teknik prioritas memiliki dampak yang positif pada efisiensi eksekusi pengujian yang sangat sulit untuk diprediksi pada situasi yang konkret. Hasil dari prioritas tujuan pengujian adalah bergantung pada kriteria kecukupan yang dipilih dan sangat bergantung pada model pengujian yang dipilih.

2.3.2. Pembentukan Kasus Uji Melalui UML State Machine Diagram dengan Kriteria Kecukupan Tertentu

Offutt dkk (1999), mendeskripsikan pengukuran dan kriteria umum dalam membentuk data pengujian perangkat lunak pada level sistem dari diagram State Machine. Kriteria tersebut termasuk juga teknik dalam membentuk pengujian pada beberapa level abstraksi untuk spesifikasi, seperti: predikat transisi (transition predicate), transisi (transition) dan rangkaian transisi (sequence of transition). Penulis mencoba mendeskripsikan spesifikasi pengujian menjadi sebuah predikat aljabar yang simple dan graph spesifikasi sebagai bahasa representasi lanjut.

Ferreira dkk (2010), menyajikan penelitian pertama untuk membangun kakas bantu analisa model kecukupan (MoCAT/ Model Coverage Analysis Tool). MoCAT menganalisa dan memvisualisasikan tingkat capaian kecukupan dari diagram UML State Machine dengan deretan pengujian yang diberikan. Kakas bantu tersebut menerima masukan berupa diagram UML State Machine, serta deretan kasus uji dalam bentuk format XML dan menghasilkan sebuah diagram UML State Machine yang telah diwarnai untuk menunjukkan hasil kecukupan yang dicapai. Kakas bantu yang dibangun untuk mengidentifikasi cakupan *state* dan transisi dari deretan pengujian dengan mensimulasikan proses eksekusi nya pada seluruh diagram State Machine. Elemen yang dilintasi direpresentasikan dengan warna hijau, sebagian cakupan dengan warna kuning, dan lainnya dengan warna merah. Kakas bantu ini hanya dapat digunakan untuk menggambarkan alur transisi dan *state* yang menunjukkan kriteria kecukupan dari suatu pengujian.

Brian dkk (2004), membuat sebuah simulasi yang presisi dan menganalisa prosedur untuk menginvestigasi efektifitas biaya untuk kecukupan sekumpulan pengujian berdasarkan diagram State Machine yang meliputi: semua transisi (all transition/AT), semua pasangan transisi (all transition pairs/ATP), semua jalur pada pohon transisi (all paths in transition trees/TT) dan ketika berhubungan, predikat penuh (full predicate/FP).

2.3.3. Teknik Lain pada Pembentukan Kasus Pengujian dengan UML State Machine Diagram

Survey yang dilakukan Manuj Anggarwal dan Sangeeta Sabharwal (2012) menyebutkan bahwa ditemukan 12 penelitian yang berhubungan dengan teknik pembentukan kasus uji dengan menggunakan diagram UML State Machine. Empat buah penelitian lainnya menggunakan GA untuk membentuk alur pengujian yang mungkin.

Doungsa dkk (2007), mengusulkan sebuah pendekatan untuk membentuk data uji dari diagram State Machine menggunakan GA. Pada penelitian ini, digunakan kriteria kecukupan untuk masing-masing transisi yang terjadi pada masing-masing level agar dapat mengukur kualitas dari masing-masing data pengujian yang terbentuk.

Shirole dkk (2011), mentransformasikan diagram diagram UML State Machine menjadi extended finite state machines (EFSM). Kemudian EFSM tersebut dibentuk menjadi *extended control flow graph*. Dan dengan menggunakan GA, kasus uji dibentuk sesuai dengan spesifikasi kecukupan (mencakup semua definisi/*all-definition-cover*, mencakup semua jalur/*all-du path cover*).

Chevalley dan Fosse (2001), mengusulkan sebuah teknik pengujian dengan pendekatan statistika untuk membentuk kasus uji dari diagram UML State Machine. Pada penelitian ini, digunakan level kecukupan transisi sebagai kriteria pengujian. Penulis mengusulkan algoritma distribusi fungsional, yang secara signifikan meningkatkan frekuensi perubahan yang mungkin dipicu oleh transisi yang keluar. Untuk membentuk kasus uji yang otomatis pada lingkungan Rose RealTime, penulis mengusulkan untuk membuat sebuah model UML khusus yang merepresentasikan program yang diuji dengan dua buah *capsule*. Generator Capsule bertugas untuk pembentukan nilai masukan, dan Collector Capsule untuk mengumpulkan data pengukuran kecukupan transisi dan mengirimkan data tersebut pada algoritma distribusi fungsional yang diimplementasikan untuk mengidentifikasi dan menyeleksi transisi pemicu. Pada penelitian ini, penulis tidak menjelaskan tentang kriteria pengujian yang digunakan. Sebagai kondisi awal, beberapa transisi saling bergantung pada satu sama lain, dan distribusi

fungsional tersebut tidak mampu membentuk sebuah kecukupan transisi yang seimbang. Tetapi penelitian ini juga belum memvalidasi aturan berhenti untuk membatasi ukuran pengujian. Selain itu, perbandingan dengan teknik pengujian dengan pendekatan deterministik lain dalam hal untuk mengukur tingkat efektifitas biaya pengujian belum pernah dilakukan sebelumnya.

Lefticaru dan Ipate (2007) mengusulkan pendekatan untuk membentuk nilai pengujian pada setiap kemungkinan urutan metode yang diturunkan dari diagram State Machine menggunakan GA.

Offut dkk (1999), mengembangkan pendekatan untuk membentuk kasus uji yang efektif dari diagram UML State Machine tanpa adanya proses transformasi. Penulis mengembangkan sebuah kakas bantu yang dapat membentuk data uji secara otomatis dan mengintegrasikannya dengan kakas bantu Rational Rose. Kasus uji diukur pada kriteria dasar kecukupan percabangan dan dalam kaitannya dengan kemampuannya dalam mendeteksi kesalahan pada perangkat lunak.

Samuel dkk (2008), mengeksplorasi alur kontrol dan data logika pada diagram UML State Machine yang secara otomatis membentuk data uji. Hong dkk (2000), mengusulkan bahwa analisa konvensional alur data dapat dimanfaatkan untuk pembentukan kasus uji dari diagram State Machine. Penulis mengusulkan metode untuk mengubah diagram State Machine menjadi sebuah EFSM dengan cara yang konservatif.

Kim dkk (1999), melakukan penelitian yang menunjukkan bahwa teknik analisa alur kontrol dan data konvensional dapat diimplementasikan untuk membentuk data uji dari diagram UML State Machines. Diagram UML State Machine dimodelkan menjadi EFSMs, lalu dari EFSMs tersebut dimodelkan menjadi *graph* alur kontrol. Selanjutnya kasus uji dibentuk dengan menerapkan teknik analisa konvensional alur data untuk menghasilkan *graph* alur. Alur kontrol diidentifikasi dalam kaitannya pada EFSMs dan hubungannya pada rangkaian urutan pesan pada diagram UML State Machine.

Kansomkeat dan Rivepiboon (2003) mengusulkan sebuah teknik otomatisasi untuk membentuk kasus uji dari UML State Machine. Diagram State Machine dibangun menggunakan kakas bantu berupa Rational Rose. Selanjutnya, diagram tersebut ditransformasikan menjadi state pemaparan struktur hirarki yang disebut

Testing Flow Graph (TFG) dengan menggunakan metode transformasi yang diusulkan. Kasus uji dibentuk dari TFG dengan menguraikan percabangan dari node paling atas pada setiap node-node anaknya untuk mencapai suatu kecukupan state dan transisi. TFG merupakan sebuah alur dan diagram struktur yang simpel serta dapat mengurangi kompleksitas dari diagram UML State Machine. Tingkat efektifitas dari kasus uji yang terbentuk diukur melalui kemampuannya dalam mendeteksi kesalahan menggunakan analisa mutasi.

Liuying dan Zhichang (1999) menyajikan sebuah metode untuk mengotomatisasi pembentukan dan penyeleksian kasus uji dari diagram UML State Machine. Pada penelitian ini digunakan WP-Method, yang diturunkan dari W-Method dengan menggunakan kumpulan karakterisasi parsial. Diagram UML State Machine ditransformasikan menjadi Finite State Machine (FSM). Sekumpulan kasus uji dibentuk dari setiap pasang perbedaan sebuah state pada implementasi pengujian. sekumpulan kasus uji yang lain dibentuk pada pengujian transisi. Setiap state diverifikasi terlebih dahulu dan setiap transisi diuji dengan pengecekan pada setiap state dan kejadian yang dibentuk dari pemicu di dalam state itu sendiri.

Swain dkk (2012), melakukan pendekatan untuk pembentukan kasus uji otomatis dari diagram UML State Machine. Diagram UML State Machine dibentuk menjadi *state chart graph*. Untuk menyeleksi predikat, digunakan DFS (Depth First Search) traversal dari *state chart graph*. Ekspresi dari hubungan antar predikat ditransformasi menjadi predikat fungsi F. EFSM dibentuk dari kode sumber (*source code*) dari predikat yang ditransformasi melalui minimalisasi fungsi F. Metode variable alternative digunakan untuk menemukan minimum F. kasus uji dibentuk untuk setiap kondisi predikat dan disimpan untuk penggunaan selanjutnya. Kakas bantu berupa modelJunit digunakan untuk memodelkan fungsionalitas model FSM atau EFSM menjadi kelas-kelas Java. Kasus uji dibentuk dari model-model tersebut dan kriteria kecukupan yang tercapai diukur. Kakas bantu tersebut dapat menampilkan secara grafis state sumber (*source state*), state tujuan (*destination state*), dan kondisi prefix jalur (*prefix path conditions*) dari data uji.

2.4. Teknik Pengujian Statistika

Penelitian ini membandingkan teknik pengujian kotak putih pada struktur program dengan teknik pengujian kotak abu-abu pada diagram UML State Machine. Algoritma ACO digunakan pada masing-masing teknik untuk membentuk data uji yang optimal. Algoritma ACO merupakan salah satu jenis teknik yang digunakan pada teknik pengujian statistika. Teknik pengujian statistika adalah teknik pengujian yang menyusun data uji berdasarkan hasil pengambilan sampel dari distribusi probabilitas, yang didefinisikan berdasarkan domain input perangkat lunak. Pengujian statistika seperti pengujian random yang menetapkan data uji dengan *sampling vector* masukan yang berdasarkan pada distribusi probabilitas (*probability distribution*). Perbedaannya, data uji tersebut seharusnya sesuai dengan struktur dari SUT, seperti yang dilakukan pada teknik pengujian struktur. Pengujian struktur mendefinisikan kriteria yang cukup bagi data uji itu sendiri melalui struktur SUT, namun dalam pengujian statistik data uji didefinisikan dari distribusi probabilitas yang memenuhi kriteria dari struktur SUT.

Pengujian statistika mungkin dipertimbangkan sebagai sebuah perluasan dari pengujian random dimana informasi struktur dari perangkat lunak yang diuji digunakan untuk mendapatkan distribusi probabilitas yang lebih seimbang dalam terminologi untuk mencakup elemen struktur dari SUT. Tujuannya adalah untuk membuat distribusi yang dilakukan lebih efisien: “Sebuah data uji akan mendeteksi lebih banyak kesalahan dari pada sebuah data uji dengan ukuran yang sama yang dibuat dari distribusi *uniform*.”

Secara alternative, pengujian statistika mungkin disebutkan sebagai *form* dari pengujian struktural dimana *random sampling* atau sampel random dari distribusi probabilitas berarti data yang digunakan sebagai data *training* atau latih pada setiap elemen batasan. Untuk mengetahui perkembangan tentang teknik dan implementasi dalam pengujian pendekatan statistika, maka telah dilakukan survey serta pemetaan pada beberapa literatur. Literatur-literatur tersebut dikelompokkan berdasarkan teknik atau algoritma penyelesaiannya. Adapun teknik yang digunakan antara lain adalah seperti pada Tabel 2.1.

Tabel 2. 1. Pembagian Teknik Pengujian Perangkat Lunak

Methodology		Literature
Generating Test Data	Search Based	Efficient Software Verification: Statistical Testing Using Automated Search (Simon & Clark, 2010)
		A Principled Evaluation of the Effect of Directed Mutation on Search-Based Statistical Testing (Poulding dkk., 2011)
		The Seed is Strong: Seeding Strategies in Search-Based Software Testing (Fraser & Arcuri, 2012)
		Search-Based Testing of Relational Schema Integrity Constraints Across Multiple Database Management Systems (Kapfhammer dkk., 2013)
	Particle Swarm Optimization	Search based constrained test case selection using execution effort (Souza dkk., 2013)
		A variable strength interaction test suites generation strategy using Particle Swarm Optimization (Ahmed & Zamli, 2011)
	Ant Colony Optimization	Adapting ant colony optimization to generate test data for software structural testing (Mao dkk., 2015)
	Huffman Code	High-Quality Statistical Test Compression With Narrow ATE Interface (Tenentes & Kavousianos, 2013)
	Others	A Survey and Comparative Study of Statistical Tests for Identifying Differential Expression from Microarray Data (Bandyopadhyay dkk., 2014)
Generating Test Case	Cluster	Random Cluster Sampling on X-Machines Test Cases (Khan & Kausar, 2013)
		A Dynamic Test Cluster Sampling Strategy by Leveraging Execution Spectra Information (Yan dkk., 2010)
		Analysis of Test Clusters for Regression Testing (Guo dkk., 2012)
	Markov Model	Modeling and Statistical Testing of Real Time Embedded Automotive Systems by Combination of Test Models and Reference Models in MATLAB/Simulink (Siegl dkk., 2011)
		Usage Modeling through Sequence Enumeration for Automated Statistical Testing of a GUI Application (Lin dkk., 2014)
		Model Based Statistical Testing of Embedded Systems (Bohr, 2011)
	Lainnya	Efficient Reduction of Model-Based Generated Test Suites Through Test Case Pair Prioritization (Cichos & Heinze, 2010)
	MeanTest	Validation of Software Testing Experiments (Hays dkk., 2014)
		Traceability Challenge 2013: Statistical Analysis for Traceability Experiments (Hays dkk., 2013)
	Bayyesian	BAUT: A Bayesian Driven Tutoring System (Tan dkk., 2010)

2.4.1. Teknik Pembentukan Kasus Pengujian

2.4.1.1. Clustering

Teknik cluster merupakan teknik yang menentukan sekumpulan skenario pengujian dari pengelompokan sekumpulan data uji. Teknik cluster telah digunakan untuk secara efektif mengurangi besarnya data uji yang digunakan. Masing-masing cluster berisi skenario uji yang berbeda dan pemilihan cluster didasarkan pada eksekusi program. Masing-masing cluster tersebut terpilih dari nilai kemungkinan yang maksimal untuk menunjukkan kesalahan dari cluster yang sama. Data input yang akan digunakan dalam pengujian ditentukan melalui distribusi probabilitas yang maksimal dalam menentukan kesalahan suatu perangkat lunak dari masing-masing cluster yang telah ditentukan.

Yan dkk. (2010) menggunakan informasi spectra dari skenario pengujian yang dipilih sebelumnya untuk menjadi pedoman pemilihan skenario pengujian menggunakan metode *clustering*. Dalam penelitian yang dilakukan, hasilnya menunjukkan bahwa Execution-Spectra-Based Sampling (ESBS) adalah lebih baik dalam mendeteksi kesalahan dari pada strategi *sampling* pada kebanyakan kasus.

Guo dkk (2012). mengusulkan untuk mengembangkan sebuah pendekatan dalam *sample cluster* suatu pengujian yang digunakan secara otomatis untuk dapat memprediksi apakah perlu melakukan *cluster* ulang pada program yang dimodifikasi.

Imtiaz Khan dan Kausar (2013) menerapkan metode Random Cluster Sampling pada data pengujian yang ditentukan dengan pendekatan formal pada X-Machines. Untuk menunjukkan teknik yang digunakan, Imtiaz Khan dan Kausar telah menentukan sebuah skenario pengujian untuk spesifikasi X-Machines Microwave oven dan kemudian menggambarkan sampel dari sekumpulan data uji menggunakan teknik Random Cluster Sampling.

2.4.1.2. Markov Model

Markov Chain Usage Models (MCUM) merupakan dasar dalam pengujian statistika selama kurang lebih dua dekade. Metode ini terbukti efektif secara ekonomi dalam memproduksi perangkat lunak yang berkualitas tinggi.

Siegl dkk. (2011) melakukan permodelan dan pengujian statistika pada sebuah perangkat lunak yang *embedded* dan *real-time* berupa sistem *Automotive* dengan menggunakan kombinasi dari model pengujian serta model referensi di Matlab. Permodelan dilakukan dengan menggunakan Time Usage Model (TUM). TUM digunakan sebagai model pengujian dan menyediakan dasar untuk menentukan semua kemungkinan skenario pengujian. Keterangan model digunakan untuk menjadi sebuah spesifikasi yang dieksekusi. Keterangan tersebut menyediakan informasi untuk evaluasi sistem yang akan diuji. TUM itu sendiri merupakan metode yang didasarkan pada algoritma Markov Chain Usage Model (MCUM) namun diperluas lagi oleh waktu. MCUM dapat digunakan dalam pengujian statistika dengan baik. TUM menyediakan integrasi dari informasi waktu untuk pengujian dengan kebutuhan real-time.

Lin dkk. (2014) juga menggunakan MCUM dalam penelitiannya. Dalam penelitian tersebut metode MCUM diimplementasikan pada antarmuka GUI suatu perangkat lunak yaitu 'the BlackBoard Quiz Editor' menggunakan kakas bantu pengujian statistika berupa JUMBL. Hasil penelitian tersebut menunjukkan untuk ke depannya diharapkan dapat dikembangkan sebuah kakas bantu pengujian statistika yang benar-benar otomatis dalam menentukan skenario uji, eksekusi skenario uji tersebut, evaluasi dan verifikasi perangkat lunak berdasarkan hasil pengujian yang telah dilakukan.

Frank Bohr (2011) mengusulkan sebuah penelitian Model Based Statistical Testing (MBST). Dalam penelitian ini digunakan Petri nets sebagai model pengujian. Dimana Petri nets adalah sebuah model yang merepresentasikan sistem terdistribusi diskret. Petri nets yang digunakan dalam penelitian ini disebut juga dengan Discrete Deterministic and Stochastic Petri Nets (DDSPN). DDSPN dapat dipetakan pada sebuah model Markov. Model ini dapat digunakan untuk menghitung informasi tentang usaha yang dikeluarkan dalam pengujian, seperti: melakukan ekspektasi panjang kasus pengujian.

2.4.1.3. Teknik Pembentukan Kasus Pengujian Lainnya

Ada banyak metode pendekatan yang diusulkan untuk mengurangi ukuran dari deretan pengujian tersebut dengan melakukan penggabungan skenario kasus pengujian. Pada penelitian yang dilakukan Cichos dan Heinze (2010) diperkenalkan sebuah pendekatan untuk mengidentifikasi penggabungan skenario kasus pengujian yang mana penggabungan tersebut dilakukan berdasarkan kemungkinan untuk mengurangi ukuran dari kasus pengujian yang besar. Penggabungan tersebut dilakukan dengan menggunakan algoritma *similarity*. Skenario kasus pengujian digambarkan sebagai transisi path dan pada dasarnya dapat dibandingkan dengan menaksir panjang path serta banyaknya bagian path yang sama. Pendekatan inilah yang dikembangkan dalam penelitian tersebut. Sehingga dari penelitian yang dilakukan, dapat diketahui bahwa hasilnya mendekati optimal jika dibandingkan dengan pengujian random.

2.4.2. Teknik Pembentukan Data Uji

2.4.2.1. Search Based

Search-Based Software Engineering (SBSE) diformulasikan dalam pengujian untuk membantu menentukan data uji. Sebuah penelitian dilakukan untuk memformulasikan ulang SBSE menjadi lebih optimal dan efisien dalam pengujian perangkat lunak. Dalam penelitian ini, digunakan algoritma seperti *search* dan teknik *operational research* untuk mempermudah menemukan solusi yang diinginkan.

Dalam penelitian yang dilakukan Poulding dkk. (2010) algoritma pencarian digunakan untuk menentukan distribusi probabilitas yang optimal dari suatu domain data uji yang akan digunakan dalam pengujian. Dalam penelitian tersebut digunakan algoritma Stochastic Hill Climbing, yaitu algoritma yang mirip dengan algoritma Random Mutation Hill Climbing yang diusulkan oleh Forest and Mitchell.

Poulding dkk. (2011) melakukan perbaikan pada algoritma yang digunakan dalam penelitian sebelumnya. Pada penelitian ini memperbaiki beberapa hal seperti: metode pencarian dengan menggunakan 'Direct Mutation'. Selain itu, dalam penelitian ini ditambahkan sebuah feedback untuk setiap evaluasi solusi potensial yang dilakukan. Sehingga dapat diperoleh informasi tentang algoritma

yang optimal untuk tahap pengujian selanjutnya. Feedback juga berfungsi untuk secara langsung memberikan petunjuk tentang ‘Mutation Operator’ dalam algoritma ‘Direct Mutation’ yang digunakan, yaitu tentang bagian mana yang merepresentasikan perubahan terbanyak yang menghasilkan solusi yang tepat.

Sebuah penelitian lain yang dilakukan Fraser dkk. (2012) melakukan analisa empiris terhadap hal yang paling utama dalam algoritma pencarian dalam menentukan keberhasilan pengujian. Dalam penelitian tersebut diketahui bahwa penempatan data uji pada skenario uji yang tepat adalah hal utama dalam pengujian. Fraser dkk. melakukan penelitian dengan mempelajari tiga buah teknik penempatan data uji yang diimplementasikan pada konteks yang berbeda. Adapun ketiga teknik tersebut berhubungan dengan penempatan konstanta yang ditentukan dari source code (SUT) dari seluruh hasil pencarian, strategi untuk memperbaiki populasi awal dari algoritma pencarian dengan tujuan untuk memberikan perbedaan serta menentukan target optimalisasi yang tepat, dan penggunaan solusi sebelumnya (solusi manual yang dilakukan) untuk menempatkan populasi awal dari algoritma pencarian.

Skema dalam database menspesifikasikan tipe data yang akan digunakan dalam suatu perangkat lunak, bagaimana data dikelompokkan menjadi tabel-tabel, yang mana data yang benar, dan hubungan apa yang ada diantara alur data tersebut. Literatur yang ditulis Kapfhammer dkk. (2013) menguraikan serta mengevaluasi secara empiris SchemaAnalyst, yaitu sebuah metode pencarian untuk efektifitas serta efisiensi pengujian dalam hubungan batasan integritas. SchemaAnalyst dibangun untuk mendukung pengujian skema yang dapat memodelkan serta mengatasi tipe data dan batasan yang berhubungan dengan tiga representasi DBMSs, yaitu: Postgres, HSQLDb, dan SQLite. Metode yang digunakan dalam SchemaAnalyst adalah AVM (Korel’s Alternating Variable Method) yang digunakan untuk menentukan deretan pengujian. Dimana deretan pengujian tersebut secara sistematis tepat dan meniadakan batasan dalam skema. Literatur ini mendeskripsikan pendekatan pada mutasi skema.

2.4.2.2.Huffman Code

Kode Huffman (Huffman Code) adalah sebuah kode variable yang telah ditentukan dengan menggunakan codewords (teks) yang pendek untuk meng-encode frekuensi kompleks yang terjadi dan codewords yang panjang untuk menentukan satu yang terpendek. Kode Huffman digunakan untuk melakukan kompresi data berupa *codewords*. Tenentes dan Kavousianos (2013) melakukan penelitian tentang Test Data Compression (TDC) dengan menggunakan pendekatan ‘Symbol-Based’, dan ‘Linear-Based’. Dalam pendekatan ‘Symbol-Based’ digunakan algoritma Kode Huffman yang hasilnya dibandingkan dengan pendekatan ‘Linear-Based’. Hasil akhir dari penelitian ini menunjukkan bahwa algoritma Kode Huffman memiliki biaya yang rendah dengan tingkat kompresi data yang tinggi (baik).

2.4.2.3.Particle Swarm Optimization (PSO)

Luciano S. de Souza dkk, (2013) melakukan penelitian untuk menyeleksi kasus uji yang terbentuk menggunakan algoritma pencarian (*search*). Pendekatan algoritma pencarian yang dilakukan adalah berdasarkan Optimasi Sekawanan Partikel (Particle Swarm Optimization/PSO) yang lebih simpel dan efisien jika dibandingkan dengan teknik pendekatan algoritma pencarian yang pernah diusulkan sebelumnya. Penelitian ini mengimplementasikan Binary Constrained PSO (BCPSO) untuk seleksi fungsional kasus uji. Hasil dari pengujian yang dilakukan menunjukkan bahwa BCPSO memberikan hasil yang menjanjikan dan lebih baik jika dibandingkan dengan teknik dengan algoritma pencarian lain.

Bestoun S Ahmed (2011) melakukan penelitian pertama untuk membentuk jumlah variabel (Variable Strength/VS) yang berinteraksi dengan deretan kasus pengujian. Pendekatan ini dikenal dengan VS Particle Swarm test Generator (VS-PSTG). VS-PSTG mengadopsi algoritma PSO untuk mengoptimalkan pengurangan ukuran pengujian.

2.4.2.4.Ant Colony Optimization (ACO)

Kualitas dari data uji yang terbentuk akan menentukan efektifitas pengujian dan berdampak pada tingkat kualitas dan ketahanan dari perangkat lunak tersebut. Chengying Mao dkk. (2015) melakukan penelitian pada teknik pengujian struktur menggunakan pendekatan ACO untuk membentuk data uji. Hasil penelitian yang

dilakukan menyebutkan bahwa algoritma ACO lebih stabil dan data uji yang terbentuk lebih memiliki kualitas yang baik jika dibandingkan teknik yang ada seperti: algoritma Local Beam Search, Simulated Annealing, ataupun Algoritma Genetic. Algoritma ACO juga dapat dikatakan sebanding dengan algoritma PSO.

2.4.3. Teknik Pembentukan Data Uji Lainnya

Literatur yang ditulis oleh Bandyopadhyay dkk. (2014) termasuk dalam kategori lainnya pada kelompok metode pengujian berdasarkan pemilihan data uji. Karena literatur ini merangkum teknik-teknik pemilihan data uji dalam penelitian-penelitian lain.

Penelitian Bandyopadhyay dkk. (2014) tersebut adalah tentang survey pada beberapa penelitian lain dalam mengidentifikasi data yang berbentuk *microarray*. Dalam literatur ini, dilakukan survey perbedaan metode pengujian *parametric* dan *non-parametric* pada proses identifikasi data uji. Hasil dari penelitian ini adalah berupa perbandingan keuntungan dan kerugian dari masing-masing pengujian *parametric* dan *non-parametric*, jika dilihat dari distribusi data, kondisi awal seperti pemilihan sampel, distribusi asumsi, struktur varian, dan hal lain yang berhubungan dengan performa dari masing-masing metode pengujian.

2.5. Algoritma Ant Colony Optimization

Pada algoritma ACO, *Graph* yang terbentuk dinotasikan sebagai berikut: $G = (V, E)$. dimana V adalah sekumpulan node pada *graph* dan E adalah garis atau *edges*. Pada *graph* dengan node sebanyak n , dan m semut (*ants*) digunakan untuk menelusuri dan membuat jalur pada *graph* tersebut. Jumlah *pheromone trail* $\tau_{(i,j)}$ yang berhubungan dengan garis (i,j) menunjukkan sifat yang dipelajari dari node terpilih j ketika semut berada pada node i . Pada semut ke- k , misal semut itu berada pada node ke- i , lalu untuk menentukan node tetangga (*neighbor*) dari posisi sekarang untuk setiap semut dinotasikan sebagai $N_{k(i)}$. Node $N_{k(i)}$ berisi node-node yang mungkin saja dikunjungi oleh semut pada langkah selanjutnya. Pada umumnya, node yang akan dipilih oleh semut pada setiap tahapnya memiliki nilai kemungkinan sebagai berikut.

$$P_k(i,j) = \frac{\tau_{(i,j)} \cdot [\mu_{(i,j)}]^\beta}{\sum_{u \in N_k(i)} \tau_{(i,u)} \cdot [\mu_{(i,u)}]^\beta} \quad (1)$$

Dimana $\mu_{(i,j)}$ adalah informasi heuristik dan biasanya dapat ditunjukkan sebagai $1/d_{(i,j)}$, disini $d_{(i,j)}$ adalah panjang garis (i,j) yang merupakan jarak antara node i dan j . β adalah parameter yang mengontrol berat relatif (*relative weight*) *pheromone trail* dan nilai heuristik.

Pada kali pertama semua semut menyelesaikan perjalanannya, nilai *pheromone* diganti pada semua garis (jalur) pada node sebagai berikut.

$$\tau_{(i,j)} \leftarrow (1 - \alpha) \cdot \tau_{(i,j)} + \Delta\tau_{(i,j)} \quad (2)$$

Dimana $\alpha \in (0,1)$ adalah parameter *pheromone* yang salah, $\Delta\tau_{k(i,j)} = \tau_{k(i,j)}$ dan $\Delta\tau_{k(i,j)}$ merepresentasikan kualitas *pheromone* pada garis (i,j) oleh semut ke k . Secara umum didefinisikan sebagai berikut.

$$\Delta\tau_{k(i,j)} = \begin{cases} \frac{1}{L_k} & \text{if } (i,j) \in T_k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Dimana T_k adalah jalur yang dilakukan oleh semut k dan L_k adalah panjangnya. Dari definisi $\Delta\tau_{k(i,j)}$, maka dapat diketahui seberapa baik performa semut yaitu: perjalanan dengan waktu tercepat, dan nilai *pheromone* yang lebih tinggi.

Selanjutnya $\Delta\tau_{(i,j)}$ didefinisikan hanya pada semut terbaik hasil seleksi global, yaitu:

$$\Delta\tau_{(i,j)} = \begin{cases} \frac{1}{L_{gb}} & \text{if } (i,j) \in \text{global} - \text{best} - \text{tour} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Dimana L_{gb} adalah panjang terbaik dari keseluruhan perjalanan yang ditempuh oleh semut selama masa uji sampel.

Secara umum, algoritma ACO digunakan untuk masalah optimasi, khususnya pada *graph*. Pada kebanyakan kasus, data masukan pengujian dipilih berdasarkan jarak Eclidean antar data masukan lainnya. Oleh karena itu, algoritma ini perlu untuk dikembangkan lebih lanjut untuk mengatasi permasalahan pada penggunaan jarak Euclidean tersebut.

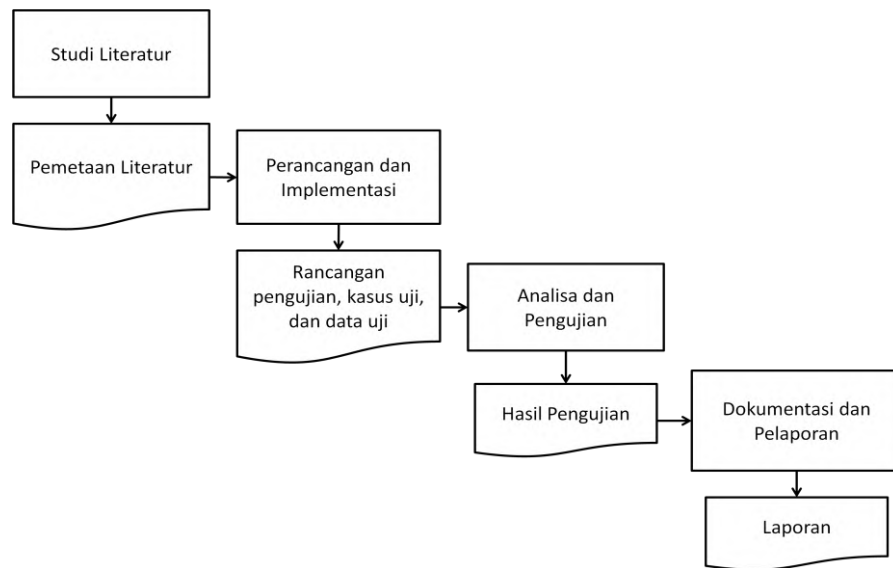
Untuk memodelkan algoritma ACO pada permasalahan pembentukan data uji pada pengujian perangkat lunak, maka n data masukan dimisalkan $X = (x_1, x_2, x_3, \dots, x_n)$ dimana data ini akan diujicobakan sebagai jarak vektor posisi dari semut pada ACO. Dengan kata lain, data uji tersebut merupakan semut-semut yang akan membentuk deretan kasus uji. Untuk masing-masing data masukan x_i diasumsikan diambil dari domain D_i ($1 \leq i \leq n$). Sehingga domain masukan untuk keseluruhan program dapat dimisalkan sebagai $D = D_1 \times D_2 \times \dots \times D_n$. Untuk masing-masing kriteria kecukupan yang diberikan, masing-masing data uji terbentuk seharusnya mampu melewati semua elemen tersebut. Pada penelitian ini digunakan kriteria kecukupan percabangan. Dimana setiap data uji yang terbentuk harus mampu mencakup semua node pada *graph*.

BAB 3

METODOLOGI PENELITIAN

3.1. Tahapan Penelitian

Metodologi penelitian terdiri dari beberapa tahap yang dapat dilihat pada Gambar 3.1



Gambar 3. 1. Tahap Metodologi Penelitian

Secara umum, luaran yang diharapkan dari penelitian ini adalah sebuah hasil perbandingan dari teknik pengujian kotak putih pada struktur kode program dan teknik pengujian kotak abu-abu pada diagram UML State Machine.

3.2. Studi Literatur

Studi literatur merupakan tahap untuk mempelajari konsep, teori, fakta dan informasi yang diperlukan dalam penelitian tesis ini. Pada suatu penelitian selalu diawali dengan proses pengkajian pustaka terkait dengan topik penelitian yang diambil. Dalam penelitian ini, referensi yang digunakan adalah literatur-literatur ilmiah yang berkaitan dengan pengujian perangkat lunak dengan pendekatan statistika serta teknik pengujian kotak abu-abu menggunakan diagram UML State Machine. Hasil dari studi literatur adalah berupa data pemetaan perkembangan dari masing-masing teknik pendekatan yang digunakan. Dari hasil tersebut, kemudian dilakukan analisa keterhubungan dengan topik terkait. Literatur-

literatur yang berhubungan dengan topik penelitian akan dipelajari lebih lanjut. Literatur tersebut antara lain adalah sebagai berikut.

- a) Literatur tentang memodelkan program pada perangkat lunak dengan menggunakan pendekatan statistika.
- b) Literatur tentang permodelan pengujian menggunakan diagram UML State Machine.
- c) Literatur tentang teknik algoritma koloni semut dalam membentuk data uji pada suatu pengujian perangkat lunak.

3.3. Perancangan dan Implementasi

Penelitian dimulai dengan penyusunan batasan, pemilihan data uji, kakas bantu yang akan digunakan, serta alur implementasi yang akan dilakukan. Batasan penelitian yang akan dilakukan berhubungan dengan bahasa pemrograman dari data set yang digunakan, banyaknya baris kode (line of code) serta batasan lain yang berhubungan dengan kakas bantu yang digunakan dan uji coba yang akan dilakukan. Batasan dalam penelitian ini telah dijelaskan pada Bab 1.6.

3.3.1 Perancangan

Data set yang digunakan dalam penelitian ini adalah delapan buah program yang terdiri dari 20 hingga 300 baris kode. Adapun data set program yang digunakan dijelaskan pada Tabel 3.1. Data set merupakan unit fungsi dari sebuah perangkat lunak. Dataset tersebut dapat diakses melalui situs web berikut: <http://tracer.lcc.uma.es/problems/testing/index.html>.

Tabel 3. 1. Data Set yang Digunakan

Program	Deskripsi
GCD	Untuk menghitung sisa hasil bagi dari dua parameter masukan a dan b. Dimana a dibagi dengan b.
insertion	Untuk mengurutkan sekumpulan bilangan dengan teknik penempatan nilai berdasarkan besar dan kecilnya.
triangle	Melakukan klasifikasi jenis-jenis segitiga.
shell	Mengurutkan sekelompok bilangan dari yang kecil hingga terbesar dengan metode shell.
heapsort	Mmengurutkan sekelompok bilangan dengan menggunakan lgoritma heapsort.
Call day	Menghitung hari dalam seminggu.
bessj	Fungsi J_n besel
Quick sort	Mengurutkan sekelompok bilangan berdasarkan algoritma quick sort.

Penelitian ini adalah membandingkan teknik pengujian kotak abu-abu dengan teknik pengujian kotak putih. Teknik kotak putih menggunakan struktur

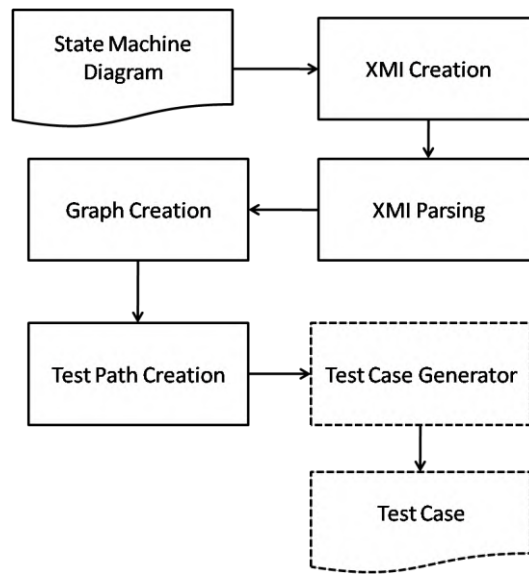
kode. Sedangkan teknik pengujian kotak abu-abu menggunakan diagram UML State Machine. Diagram UML State Machine dibangun dengan melakukan proses rekayasa balik dari kode sumber pada Tabel 3.1 tersebut. Kakas bantu yang digunakan dalam penelitian ini antara lain adalah sebagai berikut.

- a. StarUML versi 5.0.2 yaitu kakas bantu yang digunakan untuk membangun UML State Machine Diagram serta kakas bantu yang digunakan untuk mentransformasi diagram State Machine menjadi file bertipe XMI.
- b. Matlab versi 7.11 yang digunakan untuk membantu mengimplementasikan algoritma ACO dalam pembentukan kasus uji.

3.3.2 Implementasi

Pada tahap implementasi, alur yang dilakukan pada penelitian ini adalah seperti pada Gambar 3.2. Tahap implementasi dimulai dari pembuatan file XMI dengan masukan data berupa diagram UML State Machine. Selanjutnya dilanjutkan dengan menguraikan file XMI menjadi tag-tag informasi atribut dari diagram. Dari informasi atribut tersebut dibangun *graph* yang merepresentasikan alur data serta kontrol program. *Graph* tersebut selanjutnya membentuk jalur kasus pengujian yang mungkin berdasarkan kriteria kecukupan dari masing-masing percabangan. Dan Matlab digunakan untuk membantu menghitung data uji yang optimal pada proses Test Case Generator menggunakan algoritma optimisasi koloni semut.

Sebelum proses pembuatan file XMI (*XMI Creation*), terdapat proses rekayasa balik untuk membangun diagram UML State Machine dari kode sumber pada data set program Tabel 3.1. Pada proses ini pula, diagram yang dibangun divalidasi manual dengan teknik survey. Validasi manual adalah untuk mengoreksi kesalahan pada diagram yang dibangun.



Gambar 3. 2. Tahapan Proses Implementasi

Analisis hasil survey adalah berdasarkan persamaan korelasi *product moment* yang rumus persamaannya adalah sebagai berikut.

$$r_{xy} = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{(\sum x^2) - (\sum x)^2 | n(\sum y^2) - (\sum y)^2 - (\sum y)^2}} \quad (5)$$

Dimana:

- r_{xy} adalah koefisien korelasi satu butir item,
- n merupakan banyaknya subjek,
- x adalah skor satu butir soal,
- y adalah skor total.

Dasar pengambilan keputusan valid atau tidak adalah berdasarkan nilai tabel r . Jika r hitung lebih besar dari nilai r tabel, maka instrumen penelitian tersebut dinyatakan valid. Namun sebaliknya, jika nilai r hitung kurang dari nilai r tabel, maka instrumen penelitian tersebut dinyatakan tidak valid.

Penentuan kategori dari validitas instrumen adalah berdasarkan kategori validitas yang dikemukakan oleh Guilford sebagai berikut.

- 0.8-1: validitas sangat tinggi (sangat baik)
- 0.6-0.8: validitas tinggi (baik)
- 0.4-0.6: validitas sedang (cukup)
- 0.2-0.4: validitas rendah (kurang)
- 0.00-0.2: validitas sangat rendah (jelek)
- 0.00: tidak valid

Responden didefinisikan sebagai orang yang pernah belajar dan/atau menggunakan diagram UML State Machine serta pernah belajar dan/atau menggunakan bahasa pemrograman C. Diagram UML State Machine pada kasus ini tidak mendeskripsikan spesifikasi kebutuhan tertentu dari suatu perangkat lunak. Diagram ini hanya mendeskripsikan kode dari program yang digunakan sebagai data set. Penelitian ini tidak menggunakan sistem atau perangkat lunak yang utuh. Namun hanya beberapa unit fungsi dari suatu perangkat lunak. Adapun permodelan survey yang digunakan pada penelitian ini adalah sebagai berikut.

1. Choose the right/wrong on the following attributes based on diagram.

- | | | |
|---------------------|---------|---------|
| • Initial state | [right] | [wrong] |
| • Final state | [right] | [wrong] |
| • State | [right] | [wrong] |
| • Sub machine state | [right] | [wrong] |
| • Transition | [right] | [wrong] |
| • Self transition | [right] | [wrong] |
| • Junction point | [right] | [wrong] |
| • Choice point | [right] | [wrong] |
| • Synchronization | [right] | [wrong] |
| • Shallow history | [right] | [wrong] |
| • Deep history | [right] | [wrong] |
| • Flow final | [right] | [wrong] |

Attributes are based on StarUML version 5.0.2

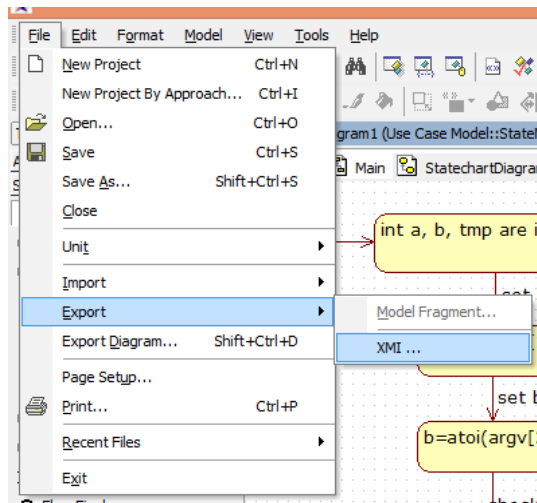
2. Does the control flow of each process in diagram appropriate with source code?
[Yes] [No]

Gambar 3. 3. Form Survey untuk Validasi Diagram Manual

3.3.2.1 XMI Creation

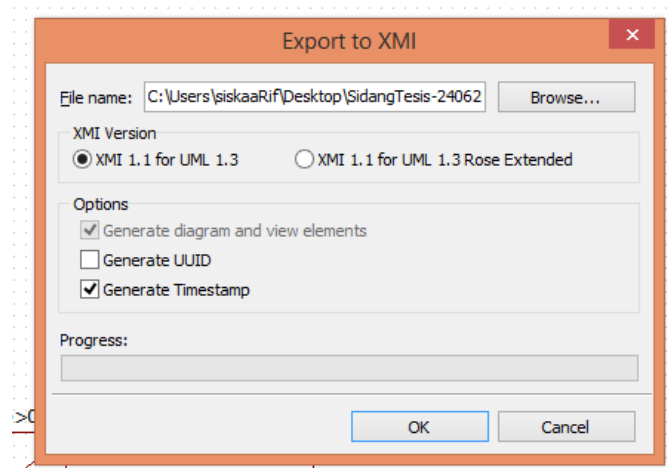
Proses *XMI Creation* adalah proses untuk membuat file XMI. File ini merupakan hasil ekspor diagram UML State Machine menggunakan Star UML. Adapun tahapan untuk melakukan ekspor pada Star UML adalah sebagai berikut.

- a) Pilih Menu [FILE], [EXPORT], [XMI] seperti pada Gambar 3.4.



Gambar 3. 4. Tahapan Menu untuk Mengekspor File XMI

- b) Pilih lokasi penyimpanan file, versi dari file XMI dan opsi lainnya seperti pada Gambar 3.5.



Gambar 3. 5. Dialog untuk Mengekspor File XMI

c) Tekan pilihan [OK]. Sehingga akan diperoleh file ‘Untitled’ seperti pada Gambar 3.6.

1.gcd_diagram	3/23/2016 3:26 PM	Microsoft Office ...	14 KB
1.gcd_kodesumber	3/23/2016 5:14 PM	Microsoft Office ...	12 KB
1.gcd_survey	4/2/2016 4:43 PM	Microsoft Office ...	13 KB
1.gcd.~ml	6/7/2016 10:27 AM	~ML File	81 KB
1.gcd	3/28/2016 12:06 PM	JPEG image	62 KB
1.gcd	6/9/2016 9:20 AM	StarUML Project	81 KB
gcd.c	5/18/2016 2:37 PM	VisualStudio.c.12.0	1 KB
Untitled	6/30/2016 12:01 PM	XML File	12 KB

Gambar 3. 6. Hasil Ekspor File XMI

3.3.2.2 XMI Parsing

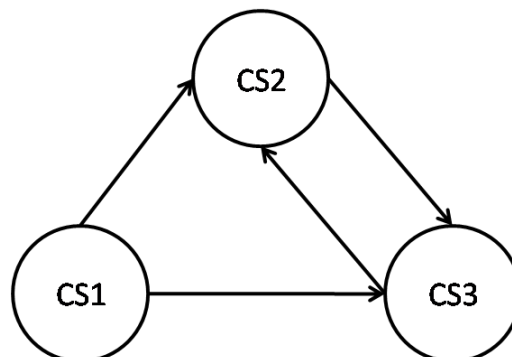
Proses ini menggunakan file XMI yang telah diekspor untuk dipisahkan tag-tag atributnya. Sehingga akan diperoleh tabel atribut sebagai berikut. Informasi pada tabel atribut akan digunakan untuk membentuk *dependency graph*.

Tabel 3. 2. Daftar Kolom pada Tabel Atribut

No	Attribute	Name	Initial
1	UML Pseudo State	initial1	PS1
2	UML Composite State	int a, b, tmp are initialized	CS1
...

3.3.2.3 Graph Creation

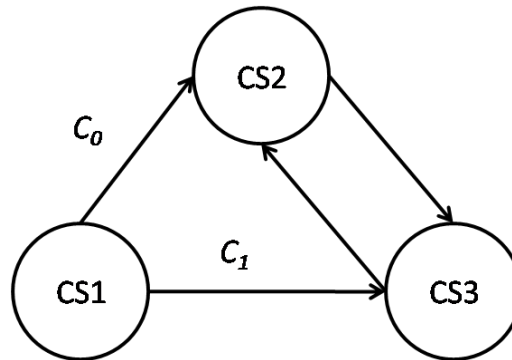
Proses ini adalah proses untuk membuat *graph*. *Graph* yang dibuat adalah berdasarkan informasi pada tabel atribut. Atribut dengan inisial CS digunakan untuk menyusun node pada *graph*. Adapun contoh *graph* yang terbentuk pada proses ini adalah seperti pada Gambar 3.7.



Gambar 3. 7. *Graph* yang Terbentuk

3.3.2.4 Test Path Creation

Proses ini membentuk jalur kasus pengujian yang mungkin berdasarkan kriteria kecukupan masing-masing percabangan. Adapun contoh dari *graph* dengan jalur kasus pengujian adalah sebagai berikut.



Gambar 3. 8. Kemungkinan Jalur Kasus Pengujian

C_0 dan C_1 merupakan kriteria kecukupan yang harus dilalui oleh masing-masing data uji. Misalkan untuk melalui C_0 maka harus memenuhi nilai variable >0 sedangkan C_1 adalah nilai variable < 0 . Sehingga berdasarkan kriteria tersebut dapat didefinisikan domain nilai variable agar bisa melewati percabangan tersebut.

3.3.2.5 Test Case Generator

Tahap terakhir adalah pembentukan kasus pengujian atau *test case generator*. Pada tahap ini digunakan kakas bantu berupa Matlab. Matlab membantu pembentukan data uji menggunakan algoritma ACO. Masukan pada Matlab adalah berupa satu file excel yang berisi deretan data uji, satu file excel yang berisi kriteria kecukupan percabangan dan parameter pengujian. Parameter pengujian adalah ditentukan berdasarkan penelitian Chengying Mao (2015). Adapun parameter tersebut antara lain adalah sebagai berikut.

- a) Parameter algoritma koloni semut:
 - a. Rmax merupakan maksimum radius jarak tetangga untuk proses transfer lokal. Nilai Rmax adalah berkisar 6-10.
 - b. Pheromone atau $\alpha = 0.3$.
 - c. Slope atau $T = 1.0$
 - d. *Adjustment coefisien* atau $\phi = 0.5$

- e. *Threshold of global random search control* $q_0 = 0.5$
- f. *Threshold of neighborhood transfer control* $p_0 = 0.3$
- b) Parameter fungsi *fitness*
 - a. Minimum level (nl-min) nilai level minimum percabangan dari *graph* yang terbentuk. Nilai ini ditentukan berdasarkan kedalaman percabangan masing-masing data set program yang digunakan.
 - b. Maksimum level (nl-max) nilai level maksimum percabangan dari *graph* yang terbentuk.
 - c. Theta (θ) koefisien untuk menghitung fungsi *fitness*. Nilai θ adalah 0.1.
 - d. Koefisien keseimbangan (λ) = 0.5
- c) maxGen adalah maksimum generasi yang menjadi penanda kapan pembentukan data uji harus berhenti.

Selain parameter tersebut terdapat jumlah semut random yang harus dipilih. Nilai ini ditentukan berdasarkan kemampuannya dalam mencakup semua percabangan (data uji yang optimal). Deretan semut random tersebut membentuk deretan pengujian (*test suites* atau ts_i) yang dilakukan. Dimana i adalah jumlah kali pengujian yang didefinisikan pada sub batasan masalah.

Penelitian ini menggunakan algoritma TDG-ACO (2015), yaitu algoritma ACO yang disempurnakan berdasarkan kasus pembentukan data uji pada pengujian perangkat lunak. Penyempurnaan tersebut salah satunya berhubungan dengan nilai fungsi *fitness*. Nilai fungsi *fitness* didefinisikan berdasarkan kedalaman level percabangan serta berat dari masing-masing predikat suatu percabangan. Persamaan dari fungsi *fitness* tersebut adalah sebagai berikut.

$$fitness(X_k) = \frac{1}{[\theta + \sum_{i=1}^s w_i \cdot f(bchi_i, X_k)]^2} \quad (6)$$

Variabel $f(bchi, X_k)$ adalah variable jarak percabangan. Jarak percabangan berupa predikat pada kondisi *if*, *else*, *case*, *for* ataupun *while*. Jarak percabangan dikategorikan berdasarkan tabel berikut.

Tabel 3. 3. Kategori Kondisi untuk Menghitung Jarak Percabangan

No	Predicate	Branch Distance
1	Boolean	If true the 0 else δ
2	$\neg a$	Negation is propagated over a
3	$a = b$	If $\text{abs}(a-b) = 0$ then 0 else $\text{abs}(a-b) + \delta$
4	$a \neq b$	If $\text{abs}(a-b) \neq 0$ then 0 else δ
5	$a < b$	If $a - b < 0$ then 0 else $\text{abs}(a-b) + \delta$
6	$a \leq b$	If $a - b \leq 0$ then 0 else $\text{abs}(a-b) + \delta$
7	$a > b$	If $b - a < 0$ then 0 else $\text{abs}(b - a) + \delta$
8	$a \geq b$	If $b - a \geq 0$ then 0 else $\text{abs}(b-a) + \delta$
9	a and b	$f(a) + f(b)$
10	a or b	$\text{Min}(f(a), f(b))$

Selain jarak percabangan, terdapat variable w atau berat. Berat dihitung berdasarkan kedalaman level dengan persamaan sebagai berikut.

$$wn(bch_i) = \frac{nl_i - nl_{min} + 1}{nl_{max} - nl_{min} + 1} \quad (7)$$

Masing-masing percabangan akan dihitung nilai beratnya berdasarkan masing-masing level kedalamannya. Selanjutnya dari masing-masing nilai tersebut akan di normalisasi melalui persamaan (8).

$$wn'(bch_i) = \frac{wn(bch_i)}{\sum_{i=1}^s wn(bch_i)} \quad (8)$$

Selain dihitung berdasarkan level ke dalamannya, berat dari suatu percabangan dihitung juga berdasarkan predikatnya. Jika predikat pada percabangan tersebut adalah sama dengan (*equality/* $==$) maka beratnya adalah 0.9, perintah logika sebagian (*partial logic order/* $<, \leq, >, \geq$) beratnya 0.6, ekspresi Boolean (*true/false*) beratnya 0.5 dan tidak sama dengan (*non-equality/* $!=$) beratnya 0.2. Jika terdapat dua buah predikat yang digunakan secara bersamaan dengan kata penghubung AND atau OR, maka aturan perhitungan beratnya adalah seperti pada persamaan (9). Pada fungsi tersebut ditulis jika digunakan kata penghubung AND, maka beratnya dapat dihitung dengan akar kuadrat dari total jumlah predikat masing-masing beratnya. Sedangkan jika menggunakan OR, maka beratnya dihitung berdasarkan nilai minimal dari keseluruhan predikat yang dipakai.

$$wp(bch_i) = \begin{cases} \sqrt{\sum_{j=1}^h w_r^2(C_j)} & \text{Jika and} \\ \min\{w_r(c_j)\}_{j=1}^h & \text{lainnya} \end{cases} \quad (9)$$

Selanjutnya berat tersebut dinormalisasi berdasarkan persamaan (10).

$$wp'(bch_i) = \frac{wp(bch_i)}{\sum_{i=1}^s wp(bch_i)} \quad (10)$$

Nilai akhir berat didefinisikan dengan persamaan 11.

$$w_i = \lambda \cdot wn'(bch_i) + (1 - \lambda) \cdot wp'(bch_i) \quad (11)$$

Selain menghitung nilai fitness untuk semut random terpilih, nilai fitness untuk keseluruhan deretan kasus pengujian (Test Suite/ TS) juga dihitung dengan menggunakan persamaan (12).

$$fitness(TS) = \frac{1}{\left[\theta + \sum_{i=1}^s w_i \cdot \min_{k=1}^m \{f(bch_i, X_k)\} \right]^2} \quad (12)$$

Selain menyempurnakan fungsi *fitness*, algoritma TDG-ACO juga menggunakan pencarian lokal dan pencarian global untuk mendapatkan hasil data uji yang optimal. Adapun *pseudocode* dari algoritma TDG-ACO adalah seperti pada Gambar 3.9. Pencarian lokal dan pencarian global adalah melakukan proses pencarian terhadap masing-masing semut random

Terdapat prosedur untuk melakukan pencarian lokal dan global. Proses ini dilakukan dengan menghitung jarak masing-masing semut dengan semut yang lain menggunakan persamaan (13). Semut yang berada di area maksimum radius akan dicek nilai fungsi *fitness* nya. Perpindahan semut dilakukan berdasarkan nilai fungsi *fitness* masing-masing semut yang berada pada area tetangga.

$$NA(X_k) = \{Y \mid \sqrt{(X_{k1} - Y_1)^2 + (X_{k1} - Y_1)^2 + \dots + (X_{k1} - Y_1)^2} \leq r \quad (13)$$

3.4. Analisis Pengujian

Penelitian ini adalah untuk membandingkan pembentukan data uji dengan algoritma ACO menggunakan UML State Machine dan kode sumber. Masing-masing kasus uji akan diujicobakan dengan teknik Dynamic Execution dari *graph* yang terbentuk. Dimana dalam teknik tersebut akan dilakukan prediksi percabangan, analisa alur data, serta spekulasi dari eksekusi yang terjadi. Hasil

dari proses ini dicatat pada tabel berikut. *Input data* merupakan parameter masukan untuk suatu data set program. *N branch* merupakan jumlah percabangan yang mampu ditelusuri oleh suatu data uji. *Time* merupakan waktu yang dibutuhkan untuk mencapai seluruh percabangan pada *graph*.

Pengukuran dilakukan dengan membandingkan nilai efektifitas dan efisiensi dari masing-masing teknik dengan mendefinisikan beberapa jenis matriks evaluasi antara lain sebagai berikut.

- a) Average Coverage (AC) atau cakupan rata-rata yaitu rata-rata jumlah percabangan yang dapat ditelusuri oleh semua data masukan pengujian pada eksekusi pengujian yang dilakukan berulang.
- b) Success Rate (SR) atau nilai kemungkinan semua percabangan yang dapat ditelusuri oleh data uji terbentuk.
- c) Average (convergence) Generation (AG) rata-rata jumlah kali beberapa kasus uji yang terbentuk untuk dapat menelusuri semua percabangan.
- d) Average Time (AT) atau nilai rata-rata waktu yaitu rata-rata waktu yang dibutuhkan untuk mencapai semua kecukupan percabangan.

Untuk mendapat nilai-nilai matriks tersebut, maka pada penelitian ini akan dilakukan 5 kali uji coba untuk masing-masing 5 generasi. Matriks evaluasi AC dan SR akan dianalisa menggunakan distribusi chi kuadrat untuk menghitung nilai *pvalue*. Kedua matriks tersebut merupakan matriks untuk mengukur cakupan percabangan serta nilai sukses kemungkinan dari percabangan yang dapat dilalui. Adapun cara menghitung nilai *pvalue* dari suatu pengujian adalah sebagai berikut.

$$\chi^2 = \sum_{i=1}^2 \frac{(x_i - x'_i)^2}{x_i} \quad (6)$$

Dimana nilai χ merupakan standar yang ditentukan untuk masing-masing teknik pengujian. Nilai standar χ adalah berdasarkan informasi total percabangan dari data set program. Sedangkan x' merupakan hasil ukur yang diperoleh ketika pengujian. Nilai signifikansi α adalah 0.05 dan nilai derajat kebebasan adalah 1. Nilai derajat kebebasan dihitung berdasarkan nilai kategori teknik pengujian perangkat lunak yang digunakan dalam penelitian dikurangi satu. Sehingga berdasarkan hasil tersebut, distribusi chi kuadrat *pvalue* berada diantara nilai 0.05-0.1. Batas atas nilai i adalah 2. Hal ini adalah berdasarkan dua buah teknik

pengujian yang dibandingkan yaitu: teknik pengujian kotak abu-abu dan teknik pengujian kotak putih. Nilai *pvalue* yang berada diantara nilai batas sebaran chi kuadrat, maka dapat disimpulkan bahwa pengujian memiliki hasil yang sesuai dengan hipotesa awal. Berdasarkan kontribusi dari penelitian ini, hipotesa awal dari penelitian ini adalah pembentukan kasus uji dengan menggunakan diagram UML State Machine (teknik pengujian kotak abu-abu) memperoleh kualitas hasil pengujian perangkat lunak yang tidak jauh berbeda atau sama jika dibandingkan dengan pembentukan kasus uji dengan menggunakan struktur kode SUT (teknik pengujian kotak putih).

[Halaman ini sengaja dikosongkan]

BAB 4

UJI COBA DAN EVALUASI

Pada bab ini akan dibagi menjadi tiga bagian yaitu implementasi penelitian, uji coba, dan analisis uji coba. Lingkungan uji coba meliputi hardware dan software yang digunakan untuk implementasi system.

4.1. Implementasi Penelitian

Penelitian ini akan dibangun dalam lingkungan pengembangan sebagai berikut:

Sistem operasi : Windows 8 32 bit
RAM : 2 GB
Processor : Intel Core i3
IDE : Matlab R2013a

4.2. Perancangan Uji Coba

Pada sub bab ini menjelaskan tentang pembagian data set, skenario uji coba, sampel proses implementasi teknik pengujian kotak abu-abu serta analisa dan evaluasi dari hasil pengujian yang dilakukan.

4.2.1. Pembagian Data Set

Dataset yang digunakan pada penelitian ini adalah dataset yang digunakan pada penelitian Chengying Mao (2015). Penelitian ini menggunakan 8 buah fungsi yang dideskripsikan pada Tabel 3.1. Tabel 4.1 menunjukkan jumlah percabangan dan baris dari masing-masing fungsi tersebut.

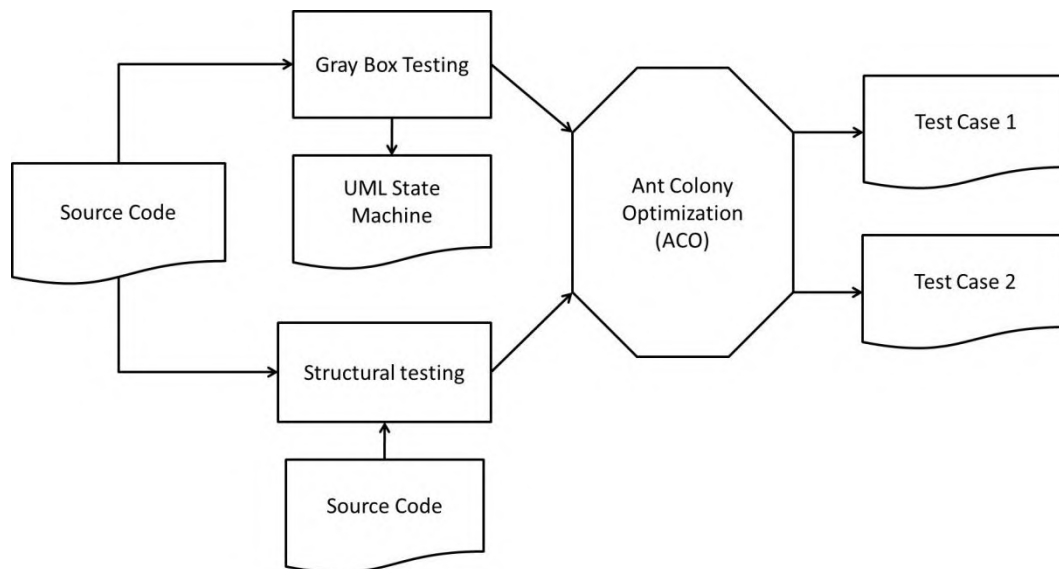
Tabel 4. 1. Data Set Fungsi yang Digunakan

Program	Percabangan	Baris
GCD	4	30
triangle	5	31
insertion	4	46
shell	6	57
heapsort	8	72
Call day	11	72
Quick sort	13	142
bessj	21	245

Tabel 4.1 menampilkan dataset program terurut berdasarkan jumlah baris dari masing-masing fungsi. Luaran dari penelitian ini adalah hasil perbandingan teknik pengujian kotak putih pada struktur kode program dengan kotak abu-abu pada diagram UML State Machine. Untuk teknik pengujian kotak abu-abu, diagram UML State Machine diperoleh dari proses rekayasa balik dari delapan fungsi yang digunakan. Kedelapan diagram UML State Machine tersebut dikoreksi dengan survey pada 18 orang responden. Diagram yang benar akan dikonversi ke dalam bentuk *graph*. Masing-masing *graph* tersebut akan ditentukan alur kasus uji untuk membantu dalam pembentukan data uji.

4.2.2. Skenario Uji Coba

Berdasarkan metode yang diusulkan, penelitian ini memiliki beberapa tahapan proses. Tahapan proses tersebut dideskripsikan pada Gambar 3.2. Skenario uji coba disusun dengan membandingkan hasil teknik pengujian kotak abu-abu dan kotak putih. Perbandingan adalah berdasarkan hasil hitung matriks pengukuran AC, SR, AG dan AT. Gambaran skenario pengujian untuk penelitian ini dapat dideskripsikan pada Gambar 4.1.



Gambar 4. 1. Skenario Uji Coba

4.2.3. Implementasi Teknik Pengujian Kotak Abu-Abu

Sub bab ini akan menjelaskan salah satu contoh implementasi teknik pengujian kotak abu-abu dari salah satu data set fungsi GCD. Gambar 4.2 mendeskripsikan fungsi GCD. Data set yang digunakan dalam penelitian ini

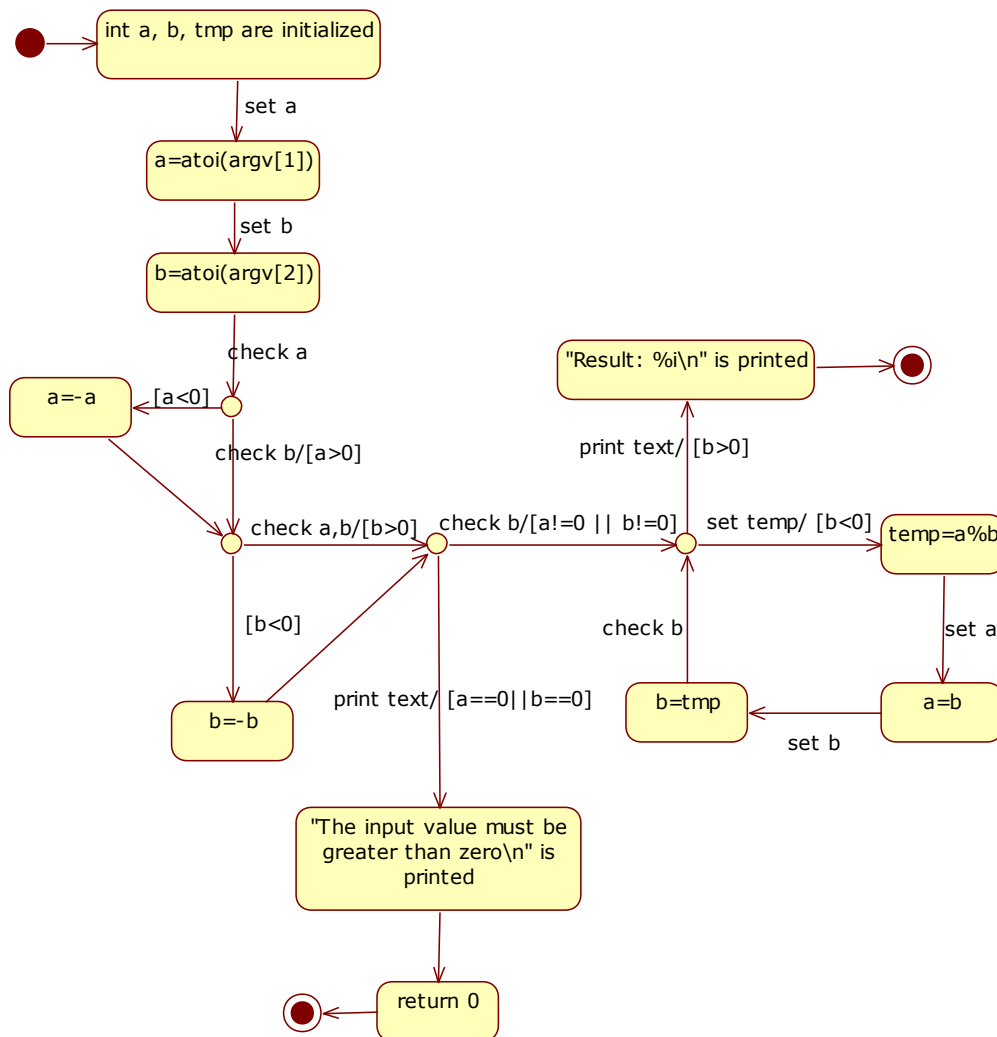
berupa unit fungsi dari sebuah perangkat lunak. Proses implementasi teknik pengujian kotak abu-abu dilakukan dengan teknik rekayasa balik dari unit fungsi tersebut menjadi diagram UML State Machine. Sub bab 3.3.2 menjelaskan teknik rekayasa balik yang digunakan dalam penelitian ini.

```
1. #include <stdio.h>
2.
3. int main (int argc, char * argv[]){
4.     int a, b, tmp;
5.
6.     a = atoi(argv[1]);
7.     b = atoi(argv[2]);
8.
9.     if (a < 0){
10.         a = -a;
11.     }
12.
13.     if (b < 0){
14.         b = -b;
15.     }
16.
17.     if (a==0 || b == 0){
18.         printf("The input values must be greater than
zero\n");
19.         scanf("%1",&a);
20.         return 0;
21.     }
22.
23.     while (b > 0){
24.         tmp = a % b;
25.         a = b;
26.         b = tmp;
27.     }
28.     printf("Result: %i\n",a);
29.     scanf("%1",&a);
30. }
```

Gambar 4. 2. Kode Sumber Data Set Program GCD

4.2.3.1. Proses UML Creation

UML Creator merupakan proses rekayasa balik untuk merubah data set program pada penelitian sebelumnya menjadi diagram UML State Machine. Proses ini dilakukan manual menggunakan kakas bantu berupa StarUML versi 5.0.2. Berdasarkan Gambar 4.1 maka diagram yang dibentuk dideskripsikan pada Gambar 4.3.



Gambar 4. 3. Diagram UML State Machine dari Fungsi GCD

4.2.3.2. Proses Diagram Validation

Diagram tersebut kemudian dikoreksi melalui teknik survey. Survey dilakukan pada 18 responden. Masing-masing responden merupakan orang yang pernah belajar dan/atau menggunakan UML State Machine Diagram serta orang yang pernah belajar dan/atau menggunakan bahasa pemrograman C. Tabel 4.2 menunjukkan hasil survey penilaian untuk diagram fungsi GCD.

Tabel 4. 2. Hasil Survey Validasi Diagram GCD

Nama Fungsi	Pertanyaan	Nilai r hitung	Nilai r tabel	Validitas	Tingkat
1. GCD	1. Initial State	1	0.37	valid	tinggi
	2. Final State	1	0.37	valid	tinggi
	3. State	0.76	0.37	valid	tinggi
	4. Sub Machine State	0.39	0.37	valid	rendah
	5. Transition	0.76	0.37	valid	tinggi
	6. Self Transition	1	0.37	valid	tinggi
	7. Junction Point	1	0.37	valid	tinggi
	8. Choice Point	1	0.37	valid	tinggi
	9. Synchronization	1	0.37	valid	tinggi
	10. Shallow History	1	0.37	valid	tinggi
	11. Deep History	1	0.37	valid	tinggi
	12. Flow Final	1	0.37	valid	tinggi
	13. Control Flow	0.72	0.37	valid	tinggi

Tabel tersebut menunjukkan hasil penilaian dari responden yang dihitung dengan menggunakan validitas konstruk. Uji validitas menggunakan rumus korelasi Product Moment. Dasar pengambilan keputusan adalah berdasarkan nilai dari tabel r.

Sehingga berdasarkan hasil penilaian pada Tabel 4.1 diagram fungsi GCD adalah benar. Penilaian validitas rendah pada Sub Machine State dikarenakan atribut ini tidak digunakan pada diagram. Dan pada umumnya responden menulis salah pada atribut diagram yang tidak digunakan tersebut.

Kasus lain jika terdapat atribut yang digunakan pada diagram dan memperoleh skor dengan kategori validitas rendah atau bahkan tidak valid, maka hal itu berarti perlu dilakukan pengecekan dan koreksi pada atribut dari diagram tersebut. Mayoritas hasil survey menunjukkan diagram adalah valid. Atribut dengan kategori validitas rendah atau tidak valid tersebut biasanya disebabkan kesalahan pada penulisan nama atribut ataupun kondisi. Fungsi *triangle* adalah salah satu fungsi yang memiliki nilai atribut tidak valid. Kesalahan pada fungsi ini adalah pada penulisan nama atribut transisi pada masing-masing percabangan.

4.2.3.3. Proses XMI Creation

Tahapan selanjutnya adalah XMI *creation*. File XMI dibentuk dengan tujuan untuk mendapatkan informasi atribut yang digunakan. Informasi atribut tersebut menyusun data pada tabel atribut. File XMI berisikan tag-tag atribut yang digunakan pada diagram beserta data id atribut tersebut.

4.2.3.4. Proses XMI Parsing

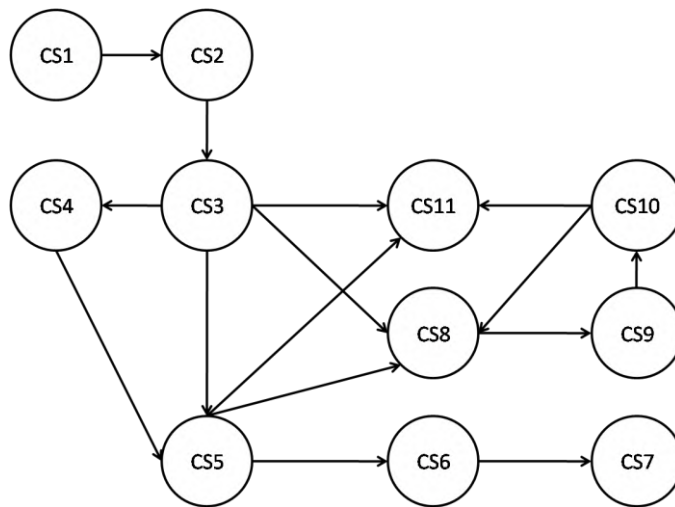
Proses ini merupakan proses untuk membentuk tabel atribut. Data informasi atribut dari file XMI disusun ke dalam bentuk tabel. Tabel 4.3 mendeskripsikan data atribut tersebut. Selanjutnya dari tabel atribut tersebut akan menjadi pedoman dalam pembentukan *graph*.

Tabel 4. 3. Atribut pada Diagram UML State Machine

No	Attribute	Name	Initial
1	UML Pseudo State	initial1	PS1
2	UML Composite State	int a, b, tmp are initialized	CS1
3	UML Composite State	a=atoi(argv[1])	CS2
4	UML Composite State	b=atoi(argv[2])	CS3
5	UMLPseudostate	Choice1	PS2
6	UMLCompositeState	a=-a	CS4
7	UMLPseudostate	Choice2	PS3
8	UMLCompositeState	b=-b	CS5
9	UMLPseudostate	Choice3	PS4
10	UMLCompositeState	print text	CS6
11	UMLCompositeState	return 0	CS7
12	UMLCompositeState	temp=a%b	CS8
13	UMLPseudostate	Choice4	PS5
14	UMLCompositeState	a=b	CS9
15	UMLCompositeState	b=tmp	CS10
16	UMLCompositeState	print text	CS11
17	UMLFinalState	FinalState1	FS1
18	UMLFinalState	FinalState2	FS2

4.2.3.5. Proses Graph Creation

Proses ini adalah proses untuk membentuk *graph*. Berdasarkan Tabel 4.3 maka *graph* yang dibentuk adalah sebagai berikut.

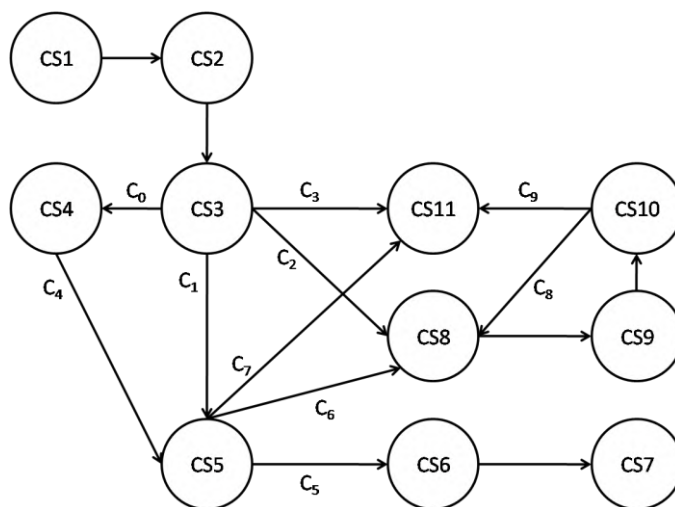


Gambar 4. 4. *Graph yang Terbentuk*

Hanya atribut Composite State yang digunakan untuk membentuk *graph*. Gambar 4.4 merupakan gambar *graph* dengan dua status keluaran yaitu CS7 jika variable yang dimasukkan adalah 0 dan CS11 untuk menampilkan hasil akhir pada fungsi GCD.

4.2.3.6. Test Path Creator

Pada proses ini alur *graph* dibentuk. Alur ini akan menjadi pedoman untuk pembentukan kemungkinan alur kasus pengujian program. C_0 , C_1 , C_2 , C_3 , C_4 , C_5 , C_6 , C_7 , C_8 , dan C_9 memiliki kriteria kecukupan untuk masing-masing percabangan. Gambar 4.5 mendeskripsikan kriteria kecukupan tersebut serta kemungkinan alur dari *graph* GCD.



Gambar 4. 5. *Alur yang Mungkin Berdasarkan Kecukupan Kriteria*

Masing-masing kemungkinan alur dapat dilalui jika dan hanya jika semua kriteria kecukupan dapat terpenuhi. Adapun kriteria kecukupan dari masing-masing alur tersebut adalah sebagai berikut.

- $C_0 = a < 0$
- $C_1 = a > 0$ dan $b < 0$
- $C_2 = a > 0$ dan $b > 0$ dan $a \neq 0$ dan $b \neq 0$ dan $b < 0$
- $C_3 = a > 0$ dan $b > 0$ dan $a \neq 0$ dan $b \neq 0$ dan $b > 0$
- $C_4 = a < 0$ dan $b < 0$
- $C_5 = b < 0$ dan $a == 0$ dan $b == 0$
- $C_6 = b < 0$ dan $a \neq 0$ dan $b \neq 0$ dan $b < 0$
- $C_7 = b < 0$ dan $a \neq 0$ dan $b \neq 0$ dan $b > 0$
- $C_8 = b < 0$
- $C_9 = b > 0$

Data uji untuk masing-masing alur pengujian dapat ditentukan berdasarkan kriteria kecukupan percabangan tersebut.

4.2.3.7. Test Case Generator

Pada proses ini pembentukan kasus uji menggunakan algoritma ACO. Parameter yang digunakan adalah sama dengan parameter yang didefinisikan sebelumnya. Adapun parameter-parameter tersebut telah dijelaskan pada bab 3.

Proses ini dimulai dengan menentukan semesta data uji yang akan digunakan. Fungsi GCD merupakan fungsi yang menggunakan dua masukan parameter a dan b. Kedua parameter tersebut memiliki tipe data integer. Sehingga untuk fungsi GCD dapat didefinisikan domain semesta a dan b adalah sebagai berikut.

$$S_a = \{a | a \in \mathbb{Z}, 0 \leq a \leq 10\} \quad (7)$$

$$S_b = \{b | b \in \mathbb{Z}, 0 \leq b \leq 10\} \quad (8)$$

Pemilihan dua domain semesta tersebut adalah berdasarkan kriteria kecukupan yang didefinisikan pada sub bab 4.2.3.6. Dengan domain semesta tersebut, maka kemungkinan kombinasi parameter a, b adalah sebagai berikut.

Tabel 4. 4. Kombinasi yang Terbentuk dari Domain Terdefinisi

(a,b)		(a,b)		(a,b)		(a,b)		(a,b)		(a,b)		(a,b)		(a,b)		(a,b)		(a,b)		(a,b)	
0	0	1	0	2	0	3	0	4	0	5	0	6	0	7	0	8	0	9	0	10	0
0	1	1	1	2	1	3	1	4	1	5	1	6	1	7	1	8	1	9	1	10	1
0	2	1	2	2	2	3	2	4	2	5	2	6	2	7	2	8	2	9	2	10	2
0	3	1	3	2	3	3	3	4	3	5	3	6	3	7	3	8	3	9	3	10	3
0	4	1	4	2	4	3	4	4	4	5	4	6	4	7	4	8	4	9	4	10	4
0	5	1	5	2	5	3	5	4	5	5	5	6	5	7	5	8	5	9	5	10	5
0	6	1	6	2	6	3	6	4	6	5	6	6	6	7	6	8	6	9	6	10	6
0	7	1	7	2	7	3	7	4	7	5	7	6	7	7	7	8	7	9	7	10	7
0	8	1	8	2	8	3	8	4	8	5	8	6	8	7	8	8	8	9	8	10	8
0	9	1	9	2	9	3	9	4	9	5	9	6	9	7	9	8	9	9	9	10	9
0	10	1	10	2	10	3	10	4	10	5	10	6	10	7	10	8	10	9	10	10	10

Selanjutnya akan dipilih 50 semut secara random. Adapun semut yang terpilih random adalah seperti pada Tabel 4.5. Ke 50 semut random tersebut akan membentuk deretan kasus pengujian pertama atau TS_1 . Jumlah 50 semut random adalah berdasarkan uji coba yang dilakukan. Fungsi GCD membutuhkan 50 semut random untuk mendapatkan hasil data uji yang maksimal. Sedangkan dataset fungsi lain, ada yang membutuhkan hingga 200 semut random. Pemilihan jumlah semut random ini adalah berdasarkan hasil capaian percabangan data uji yang terbentuk seperti yang dijelaskan pada sub bab 3.

Tabel 4. 5. 50 Semut yang Terpilih Secara Random

(a,b)		(a,b)		(a,b)		(a,b)		(a,b)	
6	4	3	9	4	5	5	2	3	1
4	4	7	2	5	10	8	1	9	2
5	6	10	3	5	3	3	3	2	7
10	9	10	7	1	2	8	6	8	5
1	2	5	2	6	9	6	7	10	2
4	8	1	7	3	3	8	5	5	1
9	8	4	6	8	5	2	9	7	6
3	4	6	2	3	10	2	5	7	10
8	8	1	6	10	7	3	1	5	5
2	7	9	2	8	9	2	3	2	6

Setelah menentukan semut secara random, selanjutnya akan dihitung nilai fungsi *fitness* dari masing-masing semut tersebut. Fungsi *fitness* didefinisikan oleh variable jarak percabangan dan berat. Seperti yang telah dijelaskan pada bab

sebelumnya, jarak antara masing-masing semut random terpilih dengan percabangannya dapat dihitung berdasarkan kategori predikat dari masing-masing percabangan. Pada fungsi GCD terdapat empat buah percabangan. Sehingga berdasarkan definisi kategori jarak percabangan pada Tabel 3.3, maka dapat dihitung jarak antara masing-masing semut random dengan percabangan pada fungsi GCD adalah seperti pada Tabel 4.6.

Tabel 4. 6. Jarak Percabangan untuk Masing-Masing Semut Random Terpilih.

	bc ₁	bc ₂	bc ₃	bc ₄		bc ₁	bc ₂	bc ₃	bc ₄
x ₁	0	0	7	0	x ₂₆	0	0	6	0
x ₂	0	0	7	0	x ₂₇	0	0	8	0
x ₃	0	0	8	0	x ₂₈	0	0	6	0
x ₄	0	0	12	0	x ₂₉	0	0	10	0
x ₅	0	0	4	0	x ₃₀	0	0	11	0
x ₆	0	0	7	0	x ₃₁	0	0	5	0
x ₇	0	0	11	0	x ₃₂	0	0	4	0
x ₈	0	0	6	0	x ₃₃	0	0	6	0
x ₉	0	0	11	0	x ₃₄	0	0	9	0
x ₁₀	0	0	5	0	x ₃₅	0	0	9	0
x ₁₁	0	0	6	0	x ₃₆	0	0	8	0
x ₁₂	0	0	5	0	x ₃₇	0	0	5	0
x ₁₃	0	0	6	0	x ₃₈	0	0	5	0
x ₁₄	0	0	10	0	x ₃₉	0	0	4	0
x ₁₅	0	0	5	0	x ₄₀	0	0	5	0
x ₁₆	0	0	4	0	x ₄₁	0	0	4	0
x ₁₇	0	0	7	0	x ₄₂	0	0	5	0
x ₁₈	0	0	5	0	x ₄₃	0	0	5	0
x ₁₉	0	0	4	0	x ₄₄	0	0	8	0
x ₂₀	0	0	5	0	x ₄₅	0	0	5	0
x ₂₁	0	0	7	0	x ₄₆	0	0	4	0
x ₂₂	0	0	8	0	x ₄₇	0	0	9	0
x ₂₃	0	0	6	0	x ₄₈	0	0	10	0
x ₂₄	0	0	4	0	x ₄₉	0	0	8	0
x ₂₅	0	0	9	0	x ₅₀	0	0	5	0

Selain menentukan jarak percabangan dari masing-masing semut random, fungsi *fitness* dapat dihitung dengan variabel berat. Adapun hasil perhitungan nilai berat adalah seperti pada Tabel 4.7.

Tabel 4. 7. Hasil Perhitungan Nilai Berat

	bc_1	bc_2	bc_3	bc_4
nl_i	2	2	3	3
$wn(bch_i)$	0.67	0.67	1	1
$wn'(bch_i)$	0.2	0.2	0.3	0.3
$wp'(bch_i)$	0.22	0.22	0.33	0.22
w_i	0.21	0.21	0.32	0.26

Setelah mendapatkan nilai berat, selanjutnya nilai *fitness* dihitung dengan menggunakan persamaan (12). *Fitness* dihitung untuk masing-masing nilai semut random terpilih. Hasil perhitungan *fitness* berdasarkan persamaan (12) adalah seperti pada Tabel 4.8.

Tabel 4. 8. Nilai *Fitness* untuk Random Ant Terpilih

X_1	0.2	X_{11}	0.3	X_{21}	0.2	X_{31}	0.4	X_{41}	0.5
X_2	0.2	X_{12}	0.4	X_{22}	0.1	X_{32}	0.5	X_{42}	0.4
X_3	0.1	X_{13}	0.3	X_{23}	0.3	X_{33}	0.3	X_{43}	0.4
X_4	0.1	X_{14}	0.1	X_{24}	0.5	X_{34}	0.1	X_{44}	0.1
X_5	0.5	X_{15}	0.4	X_{25}	0.1	X_{35}	0.1	X_{45}	0.4
X_6	0.2	X_{16}	0.5	X_{26}	0.3	X_{36}	0.1	X_{46}	0.5
X_7	0.1	X_{17}	0.2	X_{27}	0.1	X_{37}	0.4	X_{47}	0.1
X_8	0.3	X_{18}	0.4	X_{28}	0.3	X_{38}	0.4	X_{48}	0.1
X_9	0.1	X_{19}	0.5	X_{29}	0.1	X_{39}	0.5	X_{49}	0.1
X_{10}	0.4	X_{20}	0.4	X_{30}	0.1	X_{40}	0.4	X_{50}	0.4

Hasil pengujian data uji setelah memenuhi semua kecukupan percabangan untuk fungsi GCD adalah digambarkan pada Tabel 4.9.

Tabel 4. 9. Hasil Uji Coba Fungsi GCD

a	b	nBranch	Time
6	4	5	5
4	4	5	5
5	6	5	5
10	9	5	5
1	2	5	5
4	8	5	5
9	8	5	5
3	4	5	5
8	8	5	5
2	7	5	5
3	9	5	6
7	2	5	6
10	3	5	6
10	7	5	6
5	2	5	6
1	7	5	6
4	6	5	6
6	2	5	6
1	6	5	6
9	2	5	6
4	5	5	7
5	10	5	7
5	3	5	7
1	2	5	7
6	9	5	7

a	b	nBranch	Time
3	3	5	7
8	5	5	7
3	10	5	7
10	7	5	7
8	9	5	7
5	2	5	4
8	1	5	4
3	3	5	4
8	6	5	4
6	7	5	4
8	5	5	4
2	9	5	4
2	5	5	4
3	1	5	4
2	3	5	4
3	1	5	8
9	2	5	8
2	7	5	8
8	5	5	8
10	2	5	8
5	1	5	8
7	6	5	8
7	10	5	8
5	5	5	8
2	6	5	8
Average			6

4.3. Analisis Hasil

Hasil penelitian ini diukur dengan menggunakan matriks evaluasi AC, SR, AG dan AT. Berdasarkan hasil penelitian yang telah dilakukan hasil matriks tersebut adalah sebagai berikut.

Tabel 4. 10. Hasil Matrik Pengukuran pada Teknik Pengujian Kotak Abu-Abu
Menggunakan Diagram UML State Machine

NO	Program	AC	SR	AG	AT
1	GCD	100	100	5.76	6
2	Triangle	99.98	99.8	15.16	17.94
3	Insertion	100	100	9.51	11.84
4	Shell	100	100	9.58	11.18
5	Heapshort	100	100	2.01	8.6
6	Call Day	100	99.8	26.53	30.36
7	Quick Sort	99.85	99.2	17.42	96.27
8	Bessj	99.91	97.2	22.24	32.66

Tabel 4.10 menggambarkan hasil pengukuran pada teknik pengujian kotak abu-abu menggunakan diagram UML State Machine. Sedangkan hasil penelitian pada teknik pengujian kotak putih berdasarkan struktur kode adalah seperti pada Tabel 4.11.

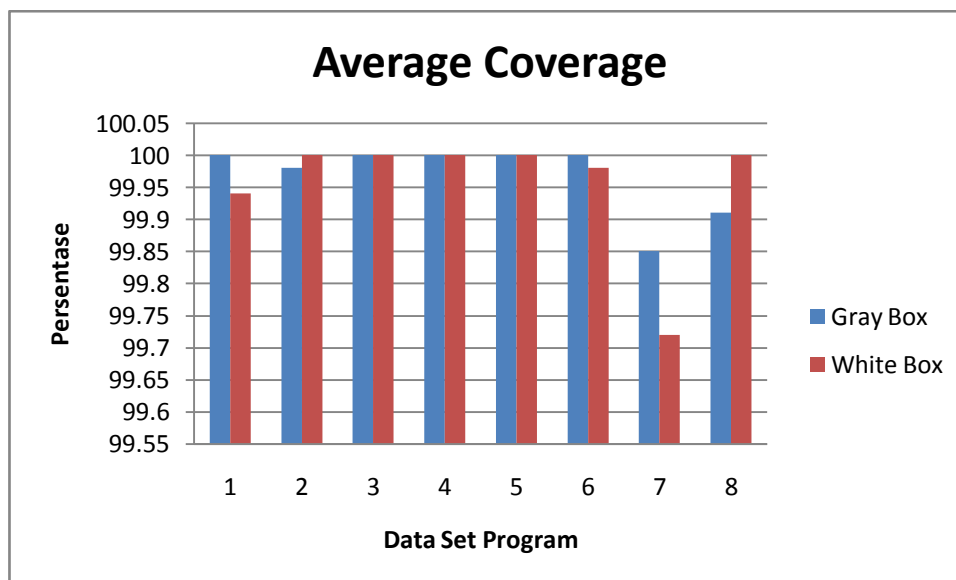
Tabel 4. 11. Hasil Matriks Pengukuran pada Teknik Pengujian Kotak Putih
Menggunakan Struktur Kode Program

NO	Program	AC	SR	AG	AT
1	GCD	99.94	99.8	5.36	6.22
2	Triangle	100	100	1.9	19.94
3	Insertion	100	100	10.37	12.84
4	Shell	100	100	8.33	11.18
5	Heapshort	100	100	5.35	10.49
6	Call Day	99.98	99.8	11.88	30.36
7	Quick Sort	99.72	99.1	12.59	96.27
8	Bessj	100	100	9.85	32.66

Hasil pengukuran dari masing-masing teknik adalah hampir mendekati dan sebanding. Kualitas data uji yang dibentuk dari teknik pengujian kotak abu-abu menggunakan diagram UML State Machine adalah sebanding dengan data uji yang dibentuk dari teknik pengujian kotak putih struktur menggunakan kode program. Hal ini dapat ditunjukkan pada Gambar 4.6. Gambar tersebut mendeskripsikan bahwa rata-rata kecukupan percabangan untuk masing-masing teknik kotak putih dan kotak abu-abu adalah tidak jauh berbeda. Hasil perbedaan yang signifikan ditunjukkan oleh data set program Bessj. Dihasilkan bahwa teknik

pengujian kotak putih adalah lebih baik cakupannya dibanding teknik kotak abu-abu. Hal ini mungkin dikarenakan oleh jumlah baris program Bessj yang mencapai 245 baris.

Gambar 4.7 menunjukkan rata-rata persentase nilai kemungkinan dari semua data uji terbentuk. Data set program Bessj memiliki nilai prosentase yang berbeda jauh antara teknik pengujian kotak putih dan kotak abu-abu. Hal ini juga dikarenakan jumlah baris kode pada program Bessj adalah lebih besar dibanding data set program lainnya. Besarnya baris kode dan jumlah percabangan berbanding lurus dengan tingkat keberhasilan cakupan yang dapat dilalui. Fungsi Bessj memiliki 24 percabangan untuk ditelusuri.



Gambar 4. 6. Grafik Perbandingan Matriks AC pada Masing-Masing Teknik

Penelitian ini menggunakan kriteria kecukupan percabangan. Kontribusi pada penelitian ini adalah untuk dapat membuktikan bahwa kualitas data uji yang terbentuk dari teknik pengujian struktur kotak putih sebanding dengan kualitas data uji yang terbentuk dari teknik pengujian kotak abu-abu. Kualitas data uji dapat diukur dari cakupan jumlah percabangan yang mampu dilalui oleh masing-masing data uji. Matriks AC dan SR merupakan dua buah matriks yang mengukur rata-rata dari kriteria tersebut. Sehingga untuk menganalisa hasil penelitian kedua matriks tersebut dianalisa berdasarkan analisa statistika menggunakan *chi kuadrat p value*.



Gambar 4. 7. Grafik Perbandingan Matriks SR pada Masing-Masing Teknik

Tabel 4.12 menunjukkan nilai hitung *pvalue* untuk matriks AC. Kolom $x_1' - x_2'$ menunjukkan selisih dari hasil pengujian masing-masing teknik. Dimana (x_1 merupakan teknik pengujian kotak abu-abu dan x_2 merupakan teknik pengujian kotak putih). Selisih tertinggi adalah pada data set program Bessj. Hal ini adalah sesuai seperti yang dideskripsikan pada Gambar 4.6. Data set program Insertion, Shell, dan Heapsort memiliki nilai *pvalue* 0. Karena pencapaian pengujian adalah sama dengan standar yang ditentukan.

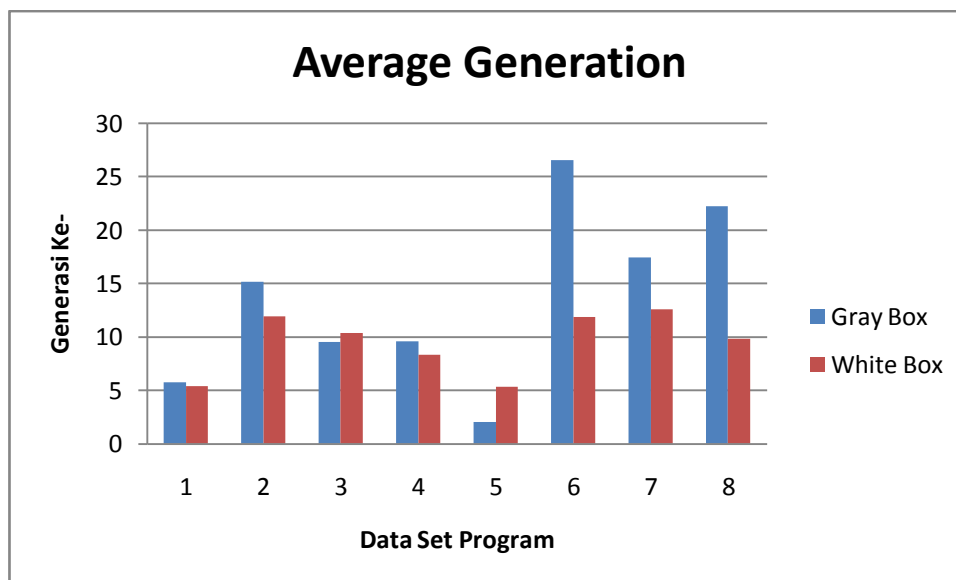
Tabel 4. 12. Nilai *pvalue* untuk Matriks AC

Program	x_1'	x_2'	$x_1' - x_2'$	<i>pvalue</i>
GCD	100	99.94	0.06	3.6E-05
Triangle	99.98	100	-0.02	4E-06
Insertion	100	100	0	0
Shell	100	100	0	0
Heapshort	100	100	0	0
Call Day	100	99.98	0.02	4E-06
Quick Sort	99.85	99.72	0.13	0.001009
Bessj	99.91	100	-0.09	8.1E-05

Tabel 4. 13. Nilai *pvalue* untuk Matriks SR

Program	x_1'	x_2'	$x_1' - x_2'$	<i>pvalue</i>
GCD	100	99.8	0.2	0.0004
Triangle	99.8	100	-0.2	0.0004
Insertion	100	100	0	0
Shell	100	100	0	0
Heapshort	100	100	0	0
Call Day	99.8	99.8	0	0.0008
Quick Sort	99.2	99.1	0.1	0.0145
Bessj	97.2	100	-2.8	0.0784

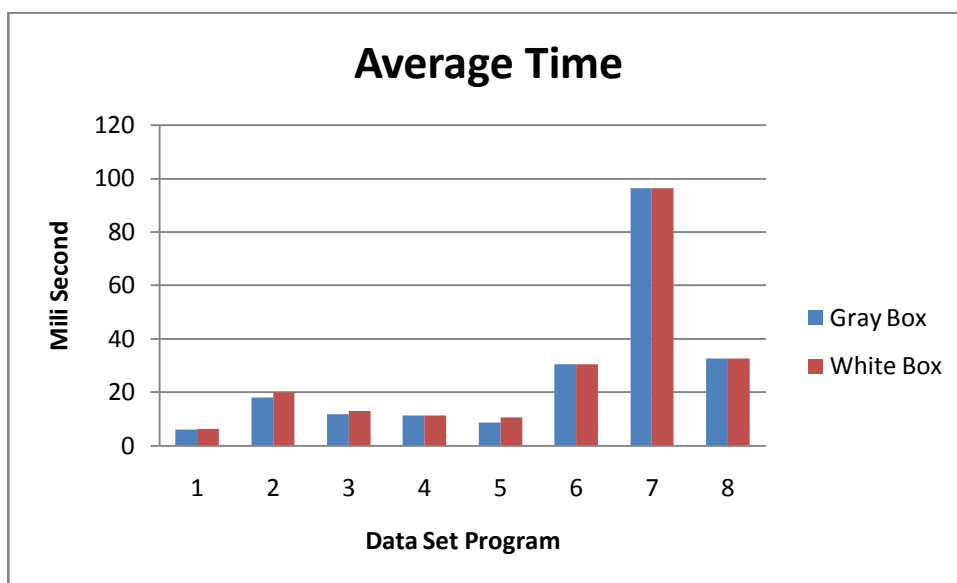
Sedangkan Tabel 4.13 mendeskripsikan analisa matriks SR. Berdasarkan tabel tersebut, nilai selisih hasil pengujian masing-masing teknik pada data set program Bessj adalah lebih besar. Hal ini adalah sama seperti yang ditunjukkan pada Gambar 4.7. Sedangkan nilai *pvalue* untuk matriks ini adalah diantara 0.05-0.1. Sehingga berdasarkan Tabel 4.12 dan Tabel 4.13 dapat disimpulkan bahwa hasil perbandingan dari teknik pengujian kotak abu-abu menggunakan diagram UML State Machine adalah tidak jauh berbeda dengan teknik pengujian kotak putih menggunakan struktur kode program.



Gambar 4. 8. Grafik Perbandingan Matriks AG pada Masing-Masing Teknik

Rata-rata generasi yang terbentuk dari koloni semut agar data uji mampu menelusuri semua percabangan dengan menggunakan teknik pengujian kotak abu-abu adalah lebih tinggi dibandingkan teknik pengujian kotak putih. Gambar 4.8

mendeskripsikan nilai matriks AG tersebut. Hal ini menunjukkan tingkat generasi yang dibentuk pada penelitian sebelumnya yaitu teknik pengujian kotak putih adalah lebih stabil dibandingkan dengan teknik pengujian kotak abu-abu. Selain itu, teknik pengujian kotak putih dilakukan dengan 100 kali uji coba hingga parameter-parameter untuk algoritma ACO konvergen. Sedangkan teknik pengujian kotak abu-abu pada penelitian ini hanya menggunakan lima buah sampel pengujian.



Gambar 4. 9. Grafik Perbandingan Matriks AT pada Masing-Masing Teknik

Gambar 4.9 mendeskripsikan perbandingan matriks AT pada masing-masing teknik. Untuk pencapaian waktu rata-rata dari kedua teknik yang digunakan adalah sama dengan atau dapat dikatakan mendekati satu sama lainnya. Data set program Triangle, Insertion, dan Heapsort memiliki nilai AT pada teknik pengujian kotak putih lebih tinggi yaitu sekitar 1-2 *mili second*. Data set program QuickSort memiliki nilai AT paling tinggi. Hal ini disebabkan data set tersebut memiliki parameter masukan sejumlah 20 data.

[Halaman ini sengaja dikosongkan]

LAMPIRAN

LAMPIRAN 1 Pseudocode Algoritma TDG-ACO

1. Pseudocode TDG-ACO

Algorithm 1. TDG-ACO

Input: (1) the program under test (PUT), and its input variable list $X = (x_1, x_2, x_3, \dots, x_n)$;

(2) structural coverage criterion C;

(3) algorithm parameters, ie., $\alpha, \varphi, \rho_0, q_0, T, m$, and r_{\max}

(4) the maximum evolution generation maxGen

Output: test data set TS satisfying the criterion C.

Stage 1: Initialization

```
1: for  $k \rightarrow 1 : m$  do
2:   for  $i \rightarrow 1 : n$  do
3:     initialize the  $i$ -th dimension ( $\text{ant}[k].x[i]$ ) of position vector of ant  $k$ ;
4:   end for
5:   calculate the fitness  $\text{ant}[k].\text{fitness}$  of ant  $k$ ;
6:    $\text{at}[k].\tau_0 = 1, \text{ant}[k].\text{count} = 0$ ;
7:   for  $u \rightarrow 1 : m$  do
8:      $\text{ant}[k].\text{record}[u] = 0$ ;
9:   end for
10: end for
11: get the best one (gbest) from all ants' fitness;
```

Stage 2: Optimum Solution Searching

```
12: while  $\text{gen} < \text{maxGen}$  or TS does not reach full coverage of criterion C do
13:   for  $k \rightarrow 1 : m$  do
14:     apply local search to ant  $k$ , and re-calculate its fitness;
15:   end for
16:   call procedure GlobalSearch();
17:   re-acquire gbest solution;
18:   call procedure UpdatePheromone();
19:   for  $k \rightarrow 1 : m$  do
20:     decode position  $\text{ant}[k].x[1..n]$  into a test case  $t_{ck} \in \text{TS}$ ;
21:     collect the coverage information by executing program with  $t_{ck}$ ;
22:   end for
23: end while
24: return TS;
```

2. Pseudocode Prosedur globalSearch()

Procedure 1. GlobalSearch()

```
1: calculate the average fitness  $f_{\text{avg}}$  of ant colony;
2: for  $k \rightarrow 1 : m$  do
3:   if  $\text{ant}[k].\text{fitness} < f_{\text{avg}}$  and  $\text{random}(0,1) < q_0$  then
4:     randomly select a position in D for ant  $k$ ;
5:   else
6:     determine the next position ( $j$ ) of ant  $k$  according to transition probability
7:      $\text{ant}[j].\text{count}++$ ;
8:      $\text{ant}[j].\text{record}[\text{ant}[j].\text{count}] = k$ ;
9:     if  $\text{random}(0,1) < \rho_0$  then
10:      transfer to an arbitrary position in the neighbor area of ant  $j$ ;
11:     else
```

```
12:     move a little distance towards ant j;
13:   end if
14: end if
15: for i  $\rightarrow$  1: n do
16:   if ant[k].x[i] exceeds the range of search space then adjust it into the boundary;
17:     adjust it into the boundary;
18:   end if
19: end for
20: renew ant[k].fitness of ant k according to its current position;
21: end for
```

3. Pseudocode Prosedur updatePheromone()

Procedure 2. UpdatePheromone()

```
1: for k  $\rightarrow$  1: m do
2:   renew the pheromone of ant k
3:   for u  $\rightarrow$  1: ant[k].count do
4:     ant[k].count do
5:   end for
6:   ant[k].count = 0
7: end for
```

LAMPIRAN 2 Data Set yang Digunakan

1. Program GCD

```
1. #include <stdio.h>
2.
3. int main (int argc, char * argv[]){
4.     int a, b, tmp;
5.
6.     a = atoi(argv[1]);
7.     b = atoi(argv[2]);
8.
9.     if (a < 0){
10.         a = -a;
11.     }
12.
13.     if (b < 0){
14.         b = -b;
15.     }
16.
17.     if (a==0 || b == 0){
18.         printf("The input values must be greater than zero\n");
19.         scanf("%i",&a);
20.         return 0;
21.     }
22.
23.     while (b > 0){
24.         tmp = a % b;
25.         a = b;
26.         b = tmp;
27.     }
28.     printf("Result: %i\n",a);
29.     scanf("%i",&a);
30. }
```

2. Program Insertion

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. void piksrt(int n, float arr[])
5. /*Sorts an array arr[1..n] into ascending numerical order, by straight insertion. n is input; arr
6. is replaced on output by its sorted rearrangement.*/
7. {
8.     int i,j;
9.     float a;
10.
11.     for (j=2;j<=n;j++){ /*Pick out each element in turn.*/
12.         a=arr[j];
13.         i=j-1;
14.
15.         while (i > 0 && arr[i] > a) { /*Look for the place to insert it.*/
16.             arr[i+1]=arr[i];
17.             i--;
18.         }
```

```

19.
20.         arr[i+1]=a; /*Insert it.*/
21.     }
22. }
23.
24.
25. #define NELEMENTS 20
26.
27. int main (int argc, char **argv){
28.     float array[NELEMENTS+1];
29.     int i;
30.
31.     for (i=1; i <= NELEMENTS; i++)
32.     {
33.         array[i] = atof(argv[i]);
34.     }
35.
36.
37.     piksrt(NELEMENTS, array);
38.
39.     for (i=1; i <= NELEMENTS; i++)
40.     {
41.         printf("%f ",array[i]);
42.     }
43.
44.     printf("\n");
45.
46. }

```

3. Program Triangle

```

1.  #include <stdio.h>
2.
3.  int triang (int a, int b, int c)
4.  {
5.
6.      int tri;
7.
8.      if ( (a<=0) || (b<= 0) || (c<=0) ) return 4;
9.
10.     tri = 0;
11.
12.     if (a==b) tri += 1;
13.     if (a==c) tri += 2;
14.     if (b==c) tri += 3;
15.     if (tri == 0)
16.     {
17.         if ( (i+j <= k) || (j+k <= i) || (i+k <= j) )
18.             tri = 4;
19.         else
20.             tri = 1;
21.         return tri;
22.     }
23.
24.     if (tri > 3) tri = 3;
25.     else if ( (tri == 1) && (i+j > k) ) tri = 2;

```

```

26.     else if ( (tri == 2) && (i+k > j) ) tri = 2;
27.     else if ( (tri == 3) && (j+k > i) ) tri = 2;
28.     else tri = 4;
29.
30.     return tri;
31.
32. }
33.
34. int main (int argc, char *argv[])
35. {
36.     int a,b,c;
37.
38.     printf ("Write tree numbers: ");
39.     sscanf(argv[1],"%d",&a);
40.     sscanf(argv[2],"%d",&b);
41.     sscanf(argv[3],"%d",&c);
42.
43.     int t = triang(a,b,c);
44.
45.     if (t==1) printf ("Scalene Triangle\n");
46.     else if (t==2)     printf ("Isosceles Triangle\n");
47.     else if (t==3)     printf ("Equilateral Triangle\n");
48.     else if (t==4)     printf ("This is not a triangle\n");
49.
50.     return 0;
51.
52. }

```

4. Program Shell

```

1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  void shell(unsigned long n, float a[])
5.  /*Sorts an array a[] into ascending numerical order by Shell's method (diminishing increment
6.  sort). a is replaced on output by its sorted rearrangement. Normally, the argument n should
7.  be set to the size of array a, but if n is smaller than this, then only the first n elements of a
8.  are sorted. This feature is used in selip.*/
9.  {
10.     unsigned long i,j,inc;
11.     float v;
12.
13.     inc=1; /*Determine the starting increment.*/
14.     do {
15.         inc *= 3;
16.         inc++;
17.     } while (inc <= n);
18.
19.     do { /*Loop over the partial sorts.*/
20.         inc /= 3;
21.         for (i=inc+1;i<=n;i++) { /*Outer loop of straight insertion.*/
22.             v=a[i];
23.             j=i;
24.             while (a[j-inc] > v) { /*Inner loop of straight insertion.*/
25.                 a[j]=a[j-inc];
26.                 j -= inc;

```

```

27.                 if (j <= inc) break;
28.                 }
29.
30.                 a[j]=v;
31.             }
32.         } while (inc > 1);
33.     }
34.
35. #define NELEMENTS 20
36.
37. int main (int argc, char **argv)
38. {
39.     float array[NELEMENTS+1];
40.     int i;
41.
42.     for (i=1; i <= NELEMENTS; i++)
43.     {
44.         array[i] = atof(argv[i]);
45.     }
46.
47.
48.     shell(NELEMENTS, array);
49.
50.     for (i=1; i <= NELEMENTS; i++)
51.     {
52.         printf("%f ",array[i]);
53.     }
54.
55.     printf("\n");
56.
57. }

```

5. Program Heapsort

```

1. #include <stdio.h>
2. #include <stdlib.h>
3.
4.
5. void hpsort(unsigned long n, float ra[])
6. /*Sorts an array ra[1..n] into ascending numerical order using the Heapsort algorithm. n is
7. input; ra is replaced on output by its sorted rearrangement.*/
8. {
9.     unsigned long i,ir,j,l;
10.    float rra;
11.
12.    if (n < 2) return;
13.
14.    l=(n >> 1)+1;
15.    ir=n;
16.    /*The index l will be decremented from its initial value down to 1 during the "hiring" (heap
17. creation) phase. Once it reaches 1, the index ir will be decremented from its initial value
18. down to 1 during the "retirement-and-promotion" (heap selection) phase.*/
19.    for (;1;) {
20.        if (l > 1) { /*Still in hiring phase.*/
21.            rra=ra[--l];
22.

```



```

23.         else { /*In retirement-and-promotion phase.*/
24.             rra=ra[ir]; /*Clear a space at end of array.*/
25.             ra[ir]=ra[1]; /*Retire the top of the heap into it.*/
26.             --ir;
27.             if (ir == 1) { /*Done with the last promotion.*/
28.                 ra[1]=rra; /*The least competent worker of all!*/
29.                 break;
30.             }
31.         }
32.
33.         i=i; /*Whether in the hiring phase or promotion phase, we
34.             here set up to sift down element rra to its proper
35.             level.*/
36.         j=i+i;
37.         while (j <= ir) {
38.             if (j < ir && ra[j] < ra[j+1]) j++; /*Compare to the better underling.*/
39.             if (rra < ra[j]) { /*Demote rra.*/
40.                 ra[i]=ra[j];
41.                 i=j;
42.                 j <<= 1;
43.             }
44.             else break; /*Found rra's level. Terminate the sift-down.*/
45.         }
46.         ra[i]=rra; /*Put rra into its slot.*/
47.     }
48. }
49.
50. #define NELEMENTS 20
51.
52. int main (int argc, char **argv)
53. {
54.     float array[NELEMENTS+1];
55.     int i;
56.
57.     for (i=1; i <= NELEMENTS; i++)
58.     {
59.         array[i] = atof(argv[i]);
60.     }
61.
62.
63.     hpsort(NELEMENTS, array);
64.
65.     for (i=1; i <= NELEMENTS; i++)
66.     {
67.         printf("%f ",array[i]);
68.     }
69.
70.     printf("\n");
71.
72. }

```

6. Program Call Day

```
1. #include <stdio.h>
2. #include <math.h>
3. #define IGREG (15+31L*(10+12L*1582))
4.
5. void error (char * txt)
6. {
7.     fprintf (stderr,txt);
8.     exit(1);
9. }
10.
11. long julday(int mm, int id, int iyyy)
12. {
13.
14.     long jul;
15.     int ja,jy=year,jm;
16.
17.
18.     if (jy == 0) error("Error: there is no year zero\n");
19.
20.     if (jy < 0) ++jy;
21.
22.     if (month > 2) {
23.         jm=month+1;
24.     }
25.     else {
26.         --jy;
27.         jm=month+13;
28.     }
29.
30.     jul = (long) (floor(365.25*jy)+floor(30.6001*jm)+day+1720995);
31.
32.     if (day+31L*(month+12L*year) >= IGREG) {
33.         ja=(int)(0.01*jy);
34.         jul += 2-ja+(int) (0.25*ja);
35.     }
36.
37.     return jul;
38. }
39.
40. int main (int argc, char ** argv)
41. {
42.
43.     int year, month, day;
44.     int jul;
45.
46.     day = atoi(argv[1]);
47.     month = atoi(argv[2]);
48.     year = atoi(argv[3]);
49.
50.     jul = julday(month, day, year);
51.
52.
53.     jul = (jul+1) % 7;
54.
55.     if (jul == 0)
56.         printf("Sunday\n");
```

```

57.     else if (jul==1)
58.         printf("Monday\n");
59.     else if (jul==2)
60.         printf("Tuesday\n");
61.     else if (jul==3)
62.         printf("Wednesday\n");
63.     else if (jul==4)
64.         printf("Thursday\n");
65.     else if (jul==5)
66.         printf("Friday\n");
67.     else if (jul==6)
68.         printf("Saturday\n");
69.
70.     return 0;
71.
72. }

```

7. Program Bessj

```

1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  #define SWAP(a,b) temp=(a);(a)=(b);(b)=temp;
5.  #define M 7
6.  #define NSTACK 50
7.  /*Here M is the size of subarrays sorted by straight insertion and NSTACK is the required auxiliary
8.  storage.*/
9.
10. static unsigned long * lvector (long nl, long nh);
11. static void free_lvector(unsigned long *v, long nl, long nh);
12. static void nrerror(char error_text[]);
13.
14. void sort(unsigned long n, float arr[])
15. /*Sorts an array arr[1..n] into ascending numerical order using the Quicksort algorithm. n is
16. input; arr is replaced on output by its sorted rearrangement.*/
17. {
18.     unsigned long i,ir=n,j,k,l=1,*istack;
19.     int jstack=0;
20.     float a,temp;
21.
22.     istack=lvector(1,NSTACK);
23.     for (;l;) { /*Insertion sort when subarray small enough.*/
24.         if (ir-l < M) {
25.             for (j=l+1;j<=ir;j++) {
26.                 a=arr[j];
27.                 for (i=j-1;i>=l;i--) {
28.                     if (arr[i] <= a) break;
29.                     arr[i+1]=arr[i];
30.
31.                 }
32.                 arr[i+1]=a;
33.             }
34.
35.             if (jstack == 0) break;
36.             ir=istack[jstack--]; /*Pop stack and begin a new round of partitioning.*/
37.             l=istack[jstack--];

```

```

38.
39.     }
40.     else {
41.
42.         k=(l+ir) >> 1; /*Choose median of left, center, and right elements
43.                        as partitioning element a. Also
44.                        rearrange so that a[l] = a[l+1] =
45.                        a[ir].*/
46.         SWAP(arr[k],arr[l+1])
47.         if (arr[l] > arr[ir]) {
48.             SWAP(arr[l],arr[ir])
49.         }
50.         if (arr[l+1] > arr[ir]) {
51.             SWAP(arr[l+1],arr[ir])
52.         }
53.         if (arr[l] > arr[l+1]) {
54.             SWAP(arr[l],arr[l+1])
55.         }
56.         i=l+1; /*Initialize pointers for partitioning.*/
57.         j=ir;
58.         a=arr[l+1]; /*Partitioning element.*/
59.
60.         for (;1;) { /*Beginning of innermost loop.*/
61.             do i++; while (arr[i] < a); /*Scan up to find element > a.*/
62.             do j--; while (arr[j] > a); /*Scan down to find element < a.*/
63.             if (j < i) break; /*Pointers crossed. Partitioning complete.*/
64.             SWAP(arr[i],arr[j]); /*Exchange elements.*/
65.         } /*End of innermost loop.*/
66.
67.         arr[l+1]=arr[j]; /*Insert partitioning element.*/
68.         arr[j]=a;
69.         jstack += 2;
70.         /*Push pointers to larger subarray on stack, process smaller subarray
71.         immediately.*/
72.         if (jstack > NSTACK) nrerror("NSTACK too small in sort.");
73.         if (ir-i+1 >= j-l) {
74.             istack[jstack]=ir;
75.             istack[jstack-1]=i;
76.             ir=j-1;
77.         }
78.         else {
79.             istack[jstack]=j-1;
80.             istack[jstack-1]=l;
81.             l=i;
82.         }
83.     }
84.
85.     free_lvector(istack,1,NSTACK);
86. }
87.
88. #define NR_END 1
89. #define FREE_ARG char*
90.
91. unsigned long *lvector(long nl, long nh)
92. /* allocate an unsigned long vector with subscript range v[nl..nh] */
93. {

```

```

94.     unsigned long *v;
95.     int tmp;
96.     v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
97.     tmp = (int)v;
98.
99.     if (!tmp) perror("allocation failure in lvector()");
100.    return v-nl+NR_END;
101. }
102.
103. void free_lvector(unsigned long *v, long nl, long nh)
104. /* free an unsigned long vector allocated with lvector() */
105. {
106.     free((FREE_ARG) (v+nl-NR_END));
107. }
108.
109. void perror(char error_text[])
110. /* Numerical Recipes standard error handler */
111. {
112.     fprintf(stderr,"Numerical Recipes run-time error...\n");
113.     fprintf(stderr,"%s\n",error_text);
114.     fprintf(stderr,"...now exiting to system...\n");
115.     exit(1);
116. }
117.
118.
119.
120. #define NELEMENTS 20
121.
122. int main (int argc, char **argv)
123. {
124.     float array[NELEMENTS+1];
125.     int i;
126.
127.     for (i=1; i <= NELEMENTS; i++)
128.     {
129.         array[i] = atof(argv[i]);
130.     }
131.
132.
133.     sort(NELEMENTS, array);
134.
135.     for (i=1; i <= NELEMENTS; i++)
136.     {
137.         printf("%f ",array[i]);
138.     }
139.
140.     printf("\n");
141.
142. }

```

BAB 5

PENUTUP

Pada bab ini dijelaskan mengenai kesimpulan akhir yang didapat setelah melakukan serangkaian uji coba pada bab sebelumnya.

5.1. Kesimpulan

Kesimpulan yang dapat diambil dalam penelitian ini antara lain adalah sebagai berikut.

- a. Membentuk data uji dengan menggunakan diagram UML State Machine dengan pendekatan pengujian kotak abu-abu adalah dengan memodelkan diagram tersebut ke dalam *graph*. *Graph* tersebut menggambarkan alur kontrol dan data dari kode sumber program. Sehingga melalui informasi tersebut data uji dibentuk. Penelitian ini menggunakan kriteria percabangan untuk merumuskan kriteria kecukupan dalam pengujian.
- b. Pembentukan data uji menggunakan diagram UML State Machine dengan algoritma ACO adalah dengan memodelkan data uji yang dipilih secara random menjadi semut-semut yang akan menjadi sampel untuk menelusuri semua kemungkinan percabangan. Kemungkinan percabangan dihitung dengan fungsi *fitness* yang didefinisikan berdasarkan jarak serta berat. Dimana nilai jarak serta berat tersebut dihitung berdasarkan nilai level (kedalaman) dan predikat dari percabangan tersebut.
- c. Kualitas data uji yang terbentuk dengan pendekatan teknik pengujian kotak abu-abu menggunakan diagram UML State Machine adalah sebanding dengan teknik pengujian kotak putih menggunakan struktur kode sumber.

5.2. Saran

Saran yang dapat diberikan untuk pengembangan lebih lanjut penelitian ini adalah sebagai berikut.

- a. Penelitian ini melakukan teknik rekayasa balik untuk melakukan pengujian dengan pendekatan kotak abu-abu. Untuk penelitian selanjutnya disarankan untuk melakukan teknik rekayasa maju atau *forward engineering*. Teknik pengujian kotak abu-abu menggunakan bahasa spesifikasi kebutuhan seperti

pada pengujian kotak hitam dan memverifikasi kebenarannya berdasarkan struktur kode seperti pada pengujian kotak putih. Proses rekayasa balik bertujuan untuk membangun diagram UML State Machine. Diagram tersebut tidak dibangun berdasarkan kebutuhan. Namun diagram tersebut dibangun berdasarkan kode sumber yang telah ada. Sehingga dalam penelitian ini tidak ada kebutuhan yang dapat divalidasi dan verifikasi. Validasi dan verifikasi hanya berdasarkan kebenaran atau koreksi dari kesesuaian diagram dengan kode sumber.

- b. Teknik pengujian kotak abu-abu dapat dilakukan dengan menggunakan data berupa model arsitektural perangkat lunak, diagram UML ataupun Finite State Machine (model *state*). Penelitian ini menggunakan diagram UML State Machine adalah karena berdasarkan penelitian sebelumnya yang menyatakan bahwa diantara diagram UML yang ada, diagram UML State Machine lebih mampu mendeteksi banyak kesalahan dari perangkat lunak dibandingkan diagram UML lainnya. Penelitian tersebut menggunakan teknik manual untuk membentuk kasus uji berdasarkan diagram UML State Machine. Namun penelitian menggunakan spesifikasi kebutuhan lain seperti model arsitektural perangkat lunak atau diagram UML lainnya dengan pendekatan pengujian kotak abu-abu mungkin dapat memberikan kontribusi yang berbeda baik dalam segi metode memodelkannya ataupun hasil perbandingan yang diperoleh dalam membentuk kualitas kasus pengujian perangkat lunak.
- c. Penelitian ini menggunakan perhitungan optimasi pembentukan data uji dengan menggunakan algoritma ACO. Penggunaan algoritma lain selain algoritma ACO juga disarankan untuk penelitian selanjutnya. Algoritma ACO merupakan algoritma dengan pendekatan statistika yang menggunakan dasar pengambilan keputusan berdasarkan nilai distribusi kemungkinan. Teknik Soft Computing seperti logika *fuzzy* serta permodelan dengan jaringan saraf tiruan juga dapat menjadi kontribusi untuk penelitian selanjutnya.

DAFTAR PUSTAKA

- Aggarwal, M., & Sabharwal, S. (2012). Test Case Generation from UML State Machine Diagram: A Survey. *Third International Conference on Computer and Communication Technology* (pp. 133-140). Allahabad: IEEE.
- Ahmed, B. S., & Zamli, K. Z. (2011). A variable strength interaction test suites generation strategy using Particle Swarm Optimization. *Journal System of Software*, 84 (12), 2171-2185.
- Bandyopadhyay, S., Mallik, S., & Mukhopadhyay, A. (2014). A Survey and Comparative Study of Statistical Tests for Identifying Differential Expression from Microarray Data. *IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS*, 11 (1), 95-115.
- Bohr, F. (2011). Model Based Statistical Testing of Embedded Systems. *Software Testing, Verification and Validation Workshops (ICSTW)*, (pp. 18-25). Berlin.
- Briand, L., Labiche, Y., & Wang, Y. (2004). Using simulation to empirically investigate Test Coverage Criteria Based on Statechart. In IEEE (Ed.), *ICSE* (pp. 86-95). -: IEEE.
- Chevalley, P., & Thevenod-Fosse, P. (2001). Automated generation of statistical test cases from UML state diagrams. In IEEE (Ed.), *COMPSAC 2001* (pp. 205-214). Chicago, IL: IEEE.
- Cichos, H., & Heinze, T. S. (2010). Efficient Reduction of Model-Based Generated Test Suites Through Test Case Pair Prioritization. *Workshop on Model Driven Engineering, Verification and Validation*, (pp. 37-42). Oslo.
- Doungsa-ard, C., Dahal, K., Hossain, K., & Suwannasart, T. (2007). Test Data generation from UML State Machine Diagram using GAs. In IEEE (Ed.), *ICSEA* (p. 47). Cap Esterel: IEEE.
- Felderer, M., & Herrmann, A. (2015). Manual test Case Derivation from UML Activity Diagram and State Machines: A Controlled Experiment. *Information Software Technology*, 61 (-), 1-15.
- Ferreira, R., Faria, J., & Paiva, A. (2010). Test coverage analysis of UML State Machines. In IEEE (Ed.), *ICSTW* (pp. 284-289). Wahington: IEEE.

- Fraser, G., & Arcuri, A. (2012). The Seed is Strong: Seeding Strategies in Search-Based Software Testing. *Software Testing, Verification and Validation (ICST)*, (pp. 121-130). Montreal.
- Guo, B., Subramaniam, M., & Chundi, P. (2012). Analysis of Test Clusters for Regression Testing. *International Conference on Software Testing, Verification and Validation*, (pp. 736-736). Montreal.
- Hays, M., Hayes, H., Stromberg, A. J., & Bathke, A. C. (2013). Traceability Challenge 2013: Statistical Analysis for Traceability Experiments. *TEFSE*, (pp. 90-94). San Fransisco.
- Hays, M., Hayes, J. H., & Bathke, A. C. (2014). Validation of Software Testing Experiments. *IEEE International Conference on Software Testing, Verification, and Validation*, (pp. 333-342). Cleveland.
- Hong, H. S., Kim, Y. G., Cha, S. D., Bae, D. H., & Ural, H. (2000). A test sequence selection method for statecharts. *Software Testing, verification and reliability*, 10 (-), 203-227.
- Kansomkeat, S., & Rivepiboon, W. (2003). Automated-generating test case using UML statechart diagram. In ACM (Ed.), *SAICSIT* (pp. 296-300). ACM.
- Kapfhammer, G. M., McMin, P., & Wright, C. J. (2013). Search-Based Testing of Relational Schema Integrity Constraints Across Multiple Database Management Systems. *Software Testing, Verification and Validation (ICST)*, (pp. 31-40). Luembourg.
- Khan, Y. I., & Kausar, S. (2013). Random Cluster Sampling on X-Machines Test Cases. *International Conference on Information Technology: New Generations* (pp. 310-316). Las Vegas: IEEE.
- Kim, Y., Hong, H., Bae, D., & Cha, S. (1999). Test cases generation from UML state diagrams. *IEE Software Process*, 146 (4), 187-192.
- Lefticaru, R., & Ipate, F. (2007). Automatic state-based test generation using genetic algorithm. In IEEE (Ed.), *SYNASC* (pp. 188-195). Timisoara : IEEE.
- Lin, L., He, J., & Song, F. (2014). Usage Modeling through Sequence Enumeration for Automated Statistical Testing of a GUI Application. *Software Engineering and Service Science (ICSESS)*, (pp. 82-85). Beijing.

- Liuying, L., & Zhichang, Q. (1999). Test selection from UML Statecharts. In IEEE (Ed.), *Proceedings of Technology of Object-Oriented Languages and System* (pp. 273-279). IEEE.
- Mao, C., Xiao, L., Yu, X., & Chen, J. (2015). Adapting Ant Colony Optimization to Generate Test Data For Software Structural Testing. *Swarm and Evolutionary Computation*, 20 (-), 23-36.
- Marinkovic, V., Kordic, B., Popovic, M., & Pekovic, V. (2013). A Method for Creating the Operational Profile of TV/STB Device to be Used for Statistical Testing. *EUROCON*, (pp. 93-97). Zagreb.
- Offutt, J., & Abdurazik, A. (1999). Generating tests from UML Specification. In R. France, & B. Rumpe (Eds.), *The Unified Modeling Language* (1723 ed., pp. 416-429). Berlin: Springer Berlin Heidelberg.
- Pareschi, F., Rovatti, R., & Setti, G. (2012). On Statistical Tests for Randomness Included in the NIST SP800-22 Test Suite and Based on the Binomial Distribution. *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, 7 (2), 491-505.
- Park, B. (2003). Hybrid Neuro-Fuzzy Application in Short Term Freeway Traffic Volume Forecasting. *Transportation Research Record*.
- Patwa, S., & Malviya, A. K. (2010). Testability of Software System. *IJRRAS*, 5 (1), 72-75.
- Poulding, S., Clark, J. A., & Waeselynck, H. (2011). A Principled Evaluation of the Effect of Directed Mutation on Search-Based Statistical Testing. *International Conference on Software Testing, Verification and Validation Workshops*, (pp. 184-193). Berlin.
- Priya, S. S., & Sheba, P. D. (2013). TEST CASE GENERATION FROM UML MODELS – A SURVEY. *IJETAE*, 3 (1), 449-459.
- S, M., A.A, A., & D.P, M. (2011). A model based Prioritization Technique for Component based Software Retesting Using UML State Chart Diagram. In IEEE (Ed.), *Electronics Computer Technology (ICECT)* (pp. 364-368). Kanyakumari : IEEE.
- S, W. (2011). Toward impact analysis of test goal prioritization on the efficient execution of automatically generated test suites based on state machines. In

- IEEE (Ed.), *International conference on quality software* (pp. 150-155). Madrid: IEEE.
- Samuel, P., Mall, R., & Bothra, A. (2008). Automatic test case generation using UML State Diagrams. (IEEE, Ed.) *Software IET*, 2 (2), 79-93.
- Sandhu, S., & Singh, A. (2011). A Systematic Approach for Software Test Cases Generation using Gray Box Testing with UML Activity Diagrams. *International Journal of COmputer Science and Technology*, 2 (4).
- Sharma, C., Sabharwal, S., & Sibal, R. (2011). Applying genetic algorithm for Prioritization of Test Case Scenario Derived from UML Diagrams. *IJCSI*, 8 (2), 433-444.
- Shirole, M., Suthar, A., & Kumar, R. (2011). Generation of improved test cases from UML state diagram using genetic algorithm. In ACM (Ed.), *ISEC* (pp. 125-134). New York: ACM.
- Siegl, S., Hielscher, K.-S., & German, R. (2011). Modeling and Statistical Testing of Real Time Embedded Automotive Systems by Combination of Test Models and Reference Models in MATLAB/Simulink. *Systems Engineering (ICSEng)*, (pp. 180-185). Las Vegas.
- Simon, P., & Clark, J. A. (2010). Efficient Software Verification: Statistical Testing Using Automated Search. *Software Engineering*, 36 (6), 763-777.
- Souza, L. S., Prudencio, R. B., Barros, F. d., & Aranha, E. H. (2013). Search based constrained test case selection using execution effort. *Expert System with Application*, 40 (12), 4887-4896.
- Swain, R., Panthi, V., Bahera, P., & Mohapatra, D. (2012). Automatic test case generation from UML State CHart Diagram. *International Journal of Computer Application*, 42 (7), 26-36.
- Tan, S., Qian, K., Fu, X., & Bhattacharya, P. (2010). BAUT: A Bayesian Driven Tutoring System. *Seventh International COnference on Information Technology*, (pp. 476-481). Las Vegas.
- Tenentes, V., & Kavousianos, X. (2013). High-Quality Statistical Test Compression With Narrow ATE Interface. *IEE Transaction On Computer-Aided Design Of Integrated Circuits and System*, 32 (9), 1369-1382.

Yan, S., Chen, Z., Zhao, Z., Zhang, C., & Zhou, Y. (2010). A Dynamic Test Cluster Sampling Strategy by Leveraging Execution Spectra Information. *Third International Conference on Software Testing, Verification and Validation*, (pp. 147-154). Paris.

BIODATA PENULIS



Penulis, Siska Arifiani, lahir di kota Sumenep pada tanggal 10 September 1990. Penulis adalah anak pertama dari dua bersaudara dan dibesarkan di kota Sumenep, Madura, Jawa Timur.

Penulis menempuh pendidikan formal di SD Negeri Kolor II (1997-2003) SMPN 1 Sumenep (2003-2006), dan SMA Negeri 1 Sumenep (2006-2009).

Pada tahun 2009-2013, penulis melanjutkan pendidikan S1 di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur. Pada tahun 2014-2016, penulis melanjutkan pendidikan Magister S2 di jurusan yang sama, yaitu Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Di Jurusan Teknik Informatika, penulis mengambil bidang minat Rekayasa Perangkat Lunak. Penulis juga aktif dalam organisasi kemahasiswaan seperti: Himpunan Mahasiswa Teknik Computer (HMTTC), Keluarga Muslim Informatika sebagai sekretaris departemen, dan ITS Bangun Desa sebagai sekretaris departemen. Penulis juga pernah menjadi finalis Gelar Karya Mahasiswa (GEMASTIK) 2011, Program Karya Mahasiswa (PKM) didanai Dikti tahun 2012 dan 2013, serta Juara 2 pada ITS Expo tahun 2011 bidang pengembangan permainan. Penulis dapat dihubungi melalui alamat email siska.arifiani90@gmail.com